# GROUND OPERATIONS AEROSPACE LANGUAGE (GOAL)

# TEXTBOOK

(NASA-TM-X-69542) GROUND OPERATIONS AEROSPACE LANGUAGE (GOAL) TEXTBOOK (NASA) 275 p HC $15.75 CSCL 09B

N73-33129

Unclas
G3/08 19644

PREPARED BY:

Larry R. Dickison

Larry R. Dickison
Launch Vehicle, Checkout Automation
and Programming Office, LV-CAP-B

APPROVED:

Freddie R. Head

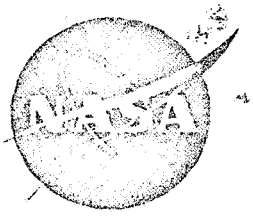
Freddie R. Head
Launch Vehicle, Checkout Automation
and Programming Office, LV-CAP-B

CONCURRENCE:

Walter J. Kapryan

Walter J. Kapryan
Director of Launch Operations

TR-1228
16 APRIL 1973



JOHN F. KENNEDY
SPACE CENTER

# GROUND OPERATIONS AEROSPACE LANGUAGE (GOAL)

TEXTBOOK

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| TR-1228 | | |

| 4. Title and Subtitle | 5. Report Date |
|---|---|
| Ground Operations Aerospace Language (GOAL) Textbook | 6. Performing Organization Code: LV-CAP |

| 7. Author(s) | 8. Performing Organization Report No. |
|---|---|
| Larry R. Dickison | |

| 9. Performing Organization Name and Address | 10. Work Unit No. |
|---|---|
| LV/Checkout Automation and Programming Office/LV-CAP | 11. Contract or Grant No. |

| 12. Sponsoring Agency Name and Address | 13. Type of Report and Period Covered: Technical Report |
|---|---|
| | 14. Sponsoring Agency Code |

15. Supplementary Notes

16. Abstract

The Textbook provides a semantical explanation accompanying a complete set of GOAL syntax diagrams, System concepts, language component interaction, and general language concepts necessary for efficient language implementation/execution.

Note: IS-DOC-3, (Printing Management Branch), is responsible for the quality of printing of this document.

| 17. KeyWords | 18. Distribution Statement |
|---|---|
| Computer Language Specification<br>Standard Test Language Specification<br>Space Shuttle Test Language Specification<br>Syntax Diagram Handbook | |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of Pages | 22. Price |
|---|---|---|---|
| Unclassified | Unclassified | | |

# PREFACE

This handbook is intended to be used as a textbook in a classroom environment. It is functionally organized to allow an instructor to cover similar statements together. Other documents developed are:

A GOAL OVERVIEW document relates the history that led to the development of the GOAL language and provides a summary of the features and capabilities of GOAL.

A GOAL SYNTAX DIAGRAMS HANDBOOK TR-1213 that contains an alphabetical arrangement of syntax diagrams used in the GOAL language.

Some pages are left blank intentionally.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

# SECTION I, FUNDAMENTALS OF GOAL

## 1.0 INTRODUCTION

GOAL, Ground Operations Aerospace Language, is a high order test language drawing from several languages in addition to NASA's experience in space vehicle testing.

The GOAL language and its associated rules will serve as the standard for test procedure specifications. Although the primary intent of GOAL is to control test procedure specification and not the associated language processors, executives, or test equipment, some suggested conventions are offered to assist in promoting uniform implementation and use.

## 1.1 SCOPE

GOAL is a test engineer oriented language, which is designed to be used as the standard procedure terminology and test programming language in performance of ground checkout operations in a space vehicle test and launch environment. It encompasses a wide range of testing, including vehicle systems and subsystems preflight checkout, ground preflight operations; such as propellant transfer, support systems verification, ground power control and monitoring, etc. The language is compatible with a wide variety of engineering design, requiring primarily command/response actions (analog and digital) to the systems to be tested. It may be used in the checkout of line replaceable units, both on-board preflight, and in the shop. It provides the flexibility to allow performance of the same procedure in both automatic and manual modes. GOAL permits a high degree of readability and retainability by providing the necessary operators required for testing, expressed in a familiar notation. Therefore, it is easily learned and understood by personnel not necessarily

skilled in programming techniques.

## 1.2 GENERAL CONCEPTS

The "statement" format was selected as the most natural form for preparing test procedures. An unrestricted free field format was adopted for the flexibility it afforded in statement positioning and paragraphing during procedure development. Judiciously used, this feature should greatly enhance the readability and the logical layout of the procedure. This format is also readily adaptable to various input mediums - computer cards, remote terminals, etc. The free field concept minimizes the importance of blanks or spaces within a statement.

### 1.2.1 Character Set

The GOAL character set is compatible with both the USA Standard Code for Information Interchange (ASCII) and the Extended Binary Coded Decimal Interchange Code (EBCDIC), allowing the individual processing systems to assign meaning to other special characters used in their system for system control and processing options.

The GOAL character set consists of:

| | | | |
|---|---|---|---|
| CAPITAL LETTERS: | A–Z | MINUS | — |
| NUMERALS | 0–9 | SLASH | / |
| SPECIAL CHARACTERS: | | SEMICOLON | ; |
| ASTERISK | * | DECIMAL POINT | . |
| BLANK | ␣ | LEFT ANGLE BRACKET | < |
| COMMA | , | RIGHT ANGLE BRACKET | > |
| CURRENCY | $ | LEFT PARENTHESIS | ( |
| EQUAL | = | RIGHT PARENTHESIS | ) |
| PLUS | + | | |

Only these characters are used in forming the language words and parameters. By convention, when writing GOAL Statements, the letter "Ø" should be slashed and the numeral "0" should not be slashed. This convention is not shown in the syntax diagrams.

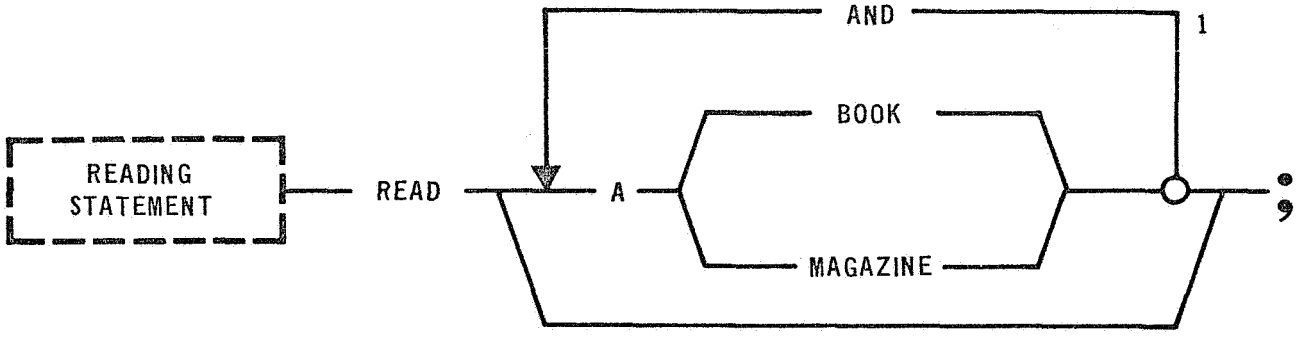
### 1.2.2 External/Internal Names

A convention was adopted that allows immediate recognition of system names and parameters that are separated because of their key role in the procedure and because by nature they are system software or hardware dependent. These items are enclosed in angle brackets, e.g., <MAIN POWER>. Items so named are centrally maintained and are made available to the Language Processor within a package called a Data Bank. These special items are called Function Designators to prevent any possible misunderstanding. All other names used in the language are enclosed in parenthesis, e.g., (PRESS SAVE). No two Names may be the same nor may any two Function Designators be the same.

Because blanks are considered insignificant when the processor is building or checking a Name or Function Designator, care must be used in selecting names that will not appear as duplicates to the Processor. For instance: (A∅B) and (AB) would be interpreted as the same Name.

## 1.3 SYNTAX DIAGRAMS

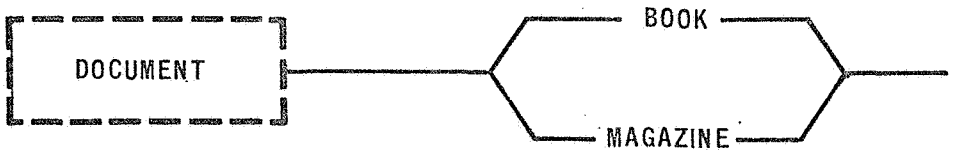
To illustrate the different allowable variations of each statement, a presentation method using syntax diagrams was selected. Syntax diagrams identify legal sequences of items in a GOAL statement, including alternate branches, optional entries, and feedback loops. The following is an example of a syntax diagram illustrating a "READING STATEMENT." Note the use of a semicolon as a statement terminator.



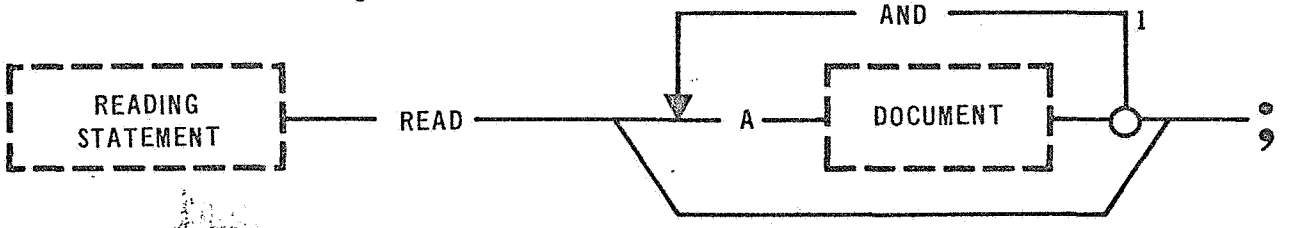

This allows any of the following sentences to be written:

```
READ;
READ A BOOK;
READ A MAGAZINE;
READ A BOOK AND A MAGAZINE;
READ A MAGAZINE AND A BOOK;
READ A BOOK AND A BOOK;
READ A MAGAZINE AND A MAGAZINE;
```

If the use of "book" and "magazine" appeared the same way in several diagrams and represented a logical grouping, then a new syntax unit could be created.




The above diagram would then become:




The dashed box represents a syntax unit. The syntax unit on the left is being defined in terms of "characters" and other syntax units.

Some basic rules for using syntax diagrams are:

- Syntax diagrams are read from left to right except for feedback loops.
- ———— is a connecting path and indicates that the insertion of blanks and/or comments is allowed.
- Capital letters must be used as shown.
- Diagonal lines are alternate forward paths.
- A bubble indicates the start of a return (feedback) path.
- A numeral at the beginning of a return path indicates the maximum number of times a path may be taken.
- A letter at the beginning of a return path indicates the number will be assigned after a system is selected.
- Syntax notes provide semantical explanation.
- GOAL statements terminate with a semicolon.
- A syntax diagram reference number is placed in each syntax unit.

To facilitate the location of any syntax diagram in this handbook, an INDEX OF SYNTAX DIAGRAMS on page 267 lists the initial words of the diagram name, the number of the diagram, and the page where it is

located. A FEEDBACK LETTERS VERSUS DIAGRAM CHART on page 265 lists the letter annotations on diagram feedbacks and the appropriate definition of each letter.

## 1.4 STATEMENT CLASSIFICATIONS

Each GOAL statement is classified as either a Declaration Statement, a Procedural Statement, or a System Statement.

### 1.4.1 Declaration Statements

A Declaration Statement is a non-test action statement that is required in an automated system to reserve storage, supply initial data conditions, or to declare the "type" of data that is valid for specified actions. Of course, in specifying manual procedures, most of declaration type information is implicit rather than explicit. Test personnel know a meter is calibrated in certain units and if the meter readings are used in computations, then care is exercised to ensure data compatibility.

GOAL declarations may be considered as either Simple Data Declarations, List Declarations, or Table Declarations. The Simple Data Declaration provides a unique name for each data entry.

The list structure is used to assign a single name to a list (one column) of data entries. The table structure is generally used to interface the test procedure with the system under test using a "rows and columns" format. Refer to Section II for more information on tables and lists.

Within each of the declaration structures the data "type" must also

be indicated. The allowable types are Numeric, Quantity, State, or Text. The four data types are defined in Section VI.

The general format of a Declaration Statement is:

DECLARE "TYPE" "STRUCTURE" "ENTRIES"

Example: DECLARE NUMERIC LIST (LIST 1) WITH 4 ENTRIES 1, 2, 3, 4;

### 1.4.2 Procedural Statements

Procedural Statements are the GOAL test action statements that specify the commands to be issued, the execution sequence, and other statements that are executed to perform a test. The general structure of a Procedural Statement is the same as a simple imperative English sentence, with the subject understood to be the computer. The minimum requirement for a Procedural Statement is a verb; however, most statements also contain an object to receive the action. An optional phrase may be used to modify the action: that is, tell when, how often, or how long to perform the action. Example:

| OPTIONAL MODIFIER | VERB | OBJECT |
|---|---|---|
| AFTER <CLOCK> IS | | |
| -3 HRS 30 MIN, | OPEN | <INLET SUPPLY VALVE> ; |

### 1.4.3 System Statements

System Statements act as directives to the Language Processor (compiler/translator) or to serve as Boundary Statements in signaling the Language Processor of the beginning, or the end, of a component.

| The general format is: | VERB | OBJECT |
|---|---|---|
| Example: | BEGIN | PROGRAM (POWER UP); |


DECLARATION STATEMENT
1
1
DECLARE DATA STATEMENT
DECLARE NUMERIC LIST STATEMENT
DECLARE NUMERIC TABLE STATEMENT
DECLARE QUANTITY LIST STATEMENT
DECLARE QUANTITY TABLE STATEMENT
DECLARE STATE LIST STATEMENT
DECLARE STATE TABLE STATEMENT
DECLARE TEXT LIST STATEMENT
DECLARE TEXT TABLE STATEMENT

# SECTION II, DECLARATION STATEMENTS

## 2.0 GENERAL

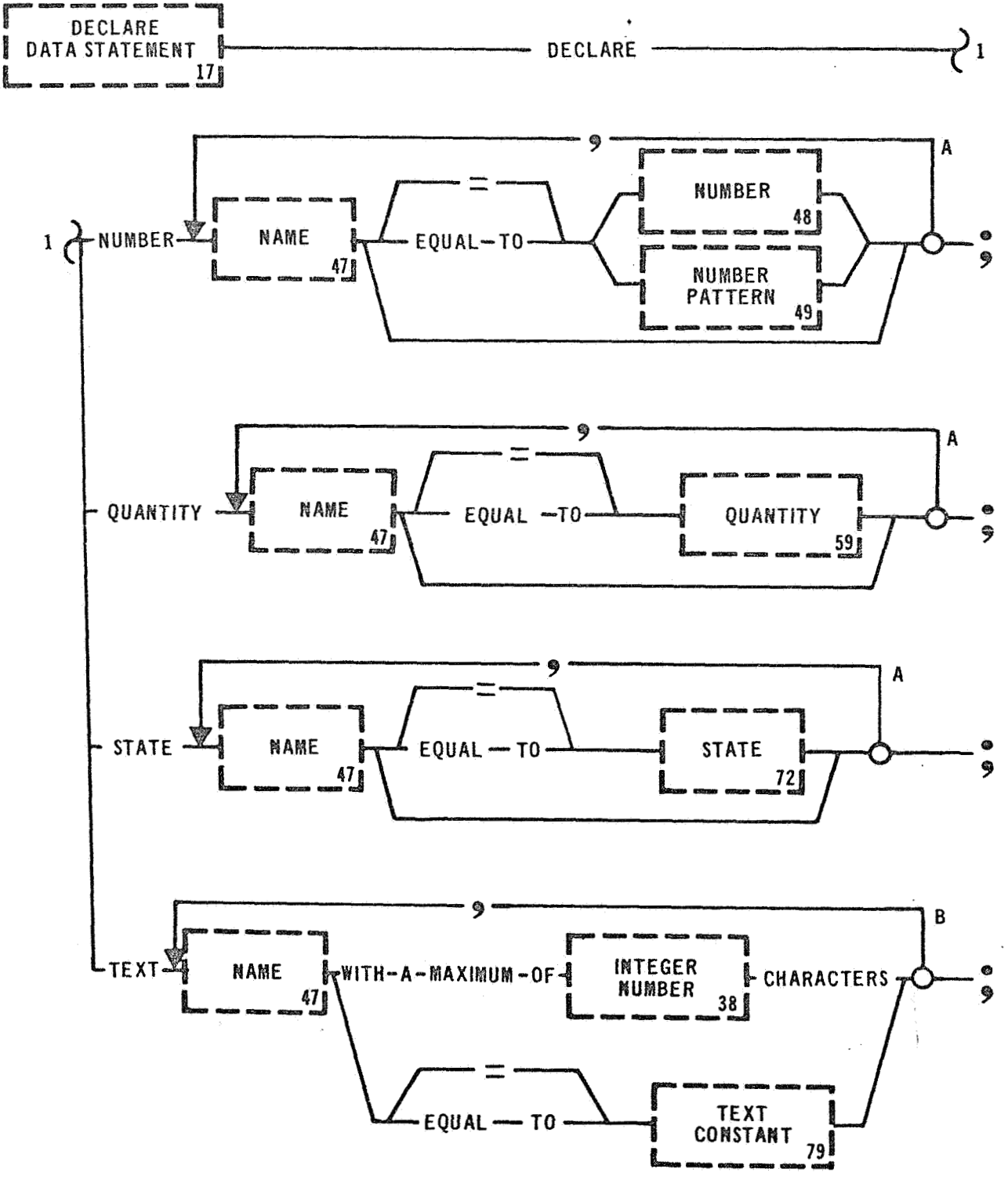
Declaration Statements are those statements required by automatic processing to instruct the system regarding the number of memory storage units to reserve for use by the procedure and to indicate the type of data that is to reside there. Initial values may be entered. GOAL has three types of statements for providing this information and for assigning a name to a collection of storage units. A single declare capability is provided for assigning a name to a single data entry. List declarations allow for the assignment of a name to a collection of related data. Table declarations allow for data columns to be used relative to one or more Function Designators. These statements are easily identified by the key word DECLARE, which is always the first word of any Declaration Statement.

## 2.1 DECLARE DATA

The simple Data Declaration Statement is used on a one-to-one, name versus data, basis.

17
REV 0

## DECLARE DATA STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

## 2.1.1 Declare Data Statement

Examples: DECLARE NUMBER (RESULTS);

DECLARE QUANTITY (OFFSET) = .5V, (PUMP PRESS);

DECLARE STATE (FLAG A) = ON;

DECLARE TEXT (ERROR MESSAGE) = (6D10 BATT VOLTAGE LOW);

The DECLARE DATA STATEMENT is a compiler directive and is used to assign data characteristics to specific names called Internal Names. Internal Names defined by a DECLARE DATA STATEMENT are internal to a test program and are not directly related to the unit under test. All Internal Names referenced by a test program must be explicitly declared.

The various data characteristics that can be assigned to an Internal Name are: Number, Quantity, State, and Text.

A Number may be declared to establish an initial value of an Internal Name. An Internal Name may be supplied an initial value of either a Decimal, Binary, Hexadecimal or Octal Number.

A Quantity may be declared to establish an initial value as a Decimal Number followed by a Dimension identifier, or as a Time Value.

A State used as an initial condition, must be either ON, OFF, OPEN, CLOSED, TRUE or FALSE.

A Text data entry may be fixed in length, i.e., a Text Constant, for use as a precanned program message. A Text variable may also be declared as a specific number of characters which may be used to reserve storage for messages that are to be inputs to a test procedure during execution. If an Internal Name is used to declare reserve storage for text data, the number of characters must be expressed as a positive integer.

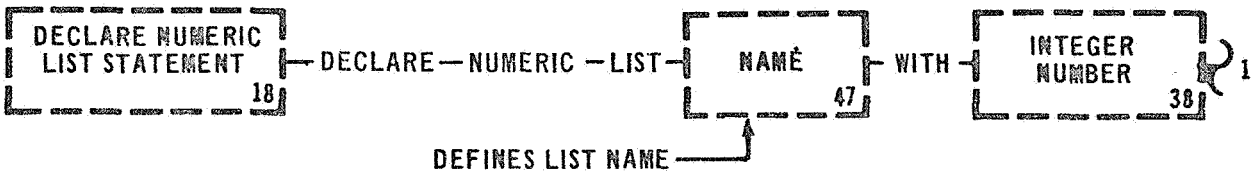
```
DECLARE NUMBER (A), (X), (Y), (Z) = X 12A3,
        (AA) = 0, (AB) EQUAL TO 1;
DECLARE QUANTITY (PRESS A) EQUAL TO 10 PSIA,
        (PRESS B) EQUAL TO 1 PSIA;
DECLARE STATE (FLAG A), (FLAG B), (FLAG C) = ON;
DECLARE TEXT (TEXT 1) WITH A MAXIMUM OF 20 CHARACTERS,
        (TEXT 2) = (REPORT PROGRAM COMPLETE);
```

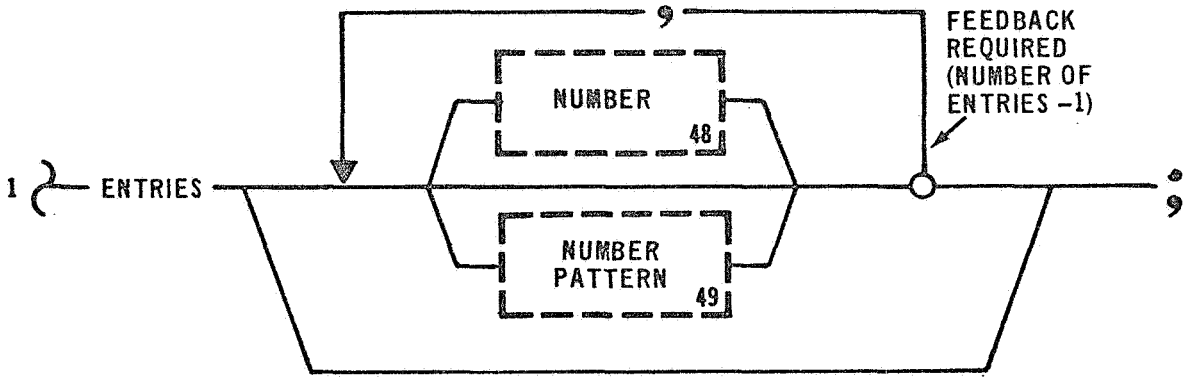
## 2.2 DECLARE LISTS

A list is used to assign a single name to a group of related data. The elements are numbered, implicitly, as they are entered, 1, 2, --- through N. Any element in a list may be referenced by using the list name and element number. In referencing, the element number may also be a numeric variable. A list may not contain Function Designators. A list is only for data. The four list statements handle the four data types: Numeric, Quantity, State, and Text. All lists require that the comma in the feedback loop be used even when the actual data is to be obtained during execution. A validity check is performed to insure that "N-1 commas" are used where "N" is the number of entries. A bypass is provided if no entries are present.

18 REV 0

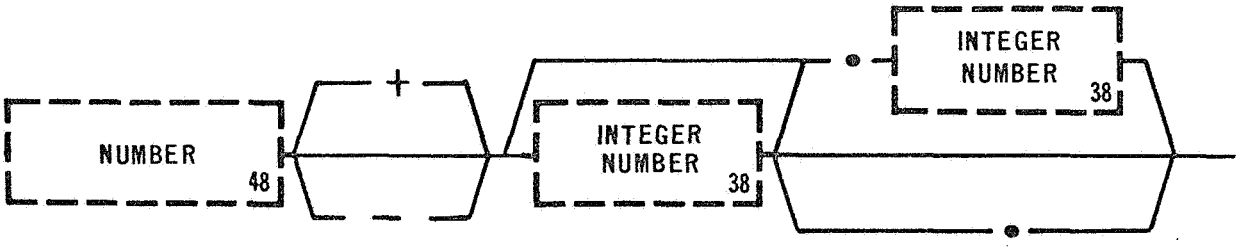
## DECLARE NUMERIC LIST STATEMENT




48 REV 1

## NUMBER



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

## 2.2.1 Declare Numeric List Statement

Examples: DECLARE NUMERIC LIST (LIST NUM) WITH 4 ENTRIES;

DECLARE NUMERIC LIST (ROOT 3) WITH 10 ENTRIES 1.000,
1.260, 1.442, 1.587, 1.710, 1.817, 1.9.3, 2.000,
2.080, 2.154;

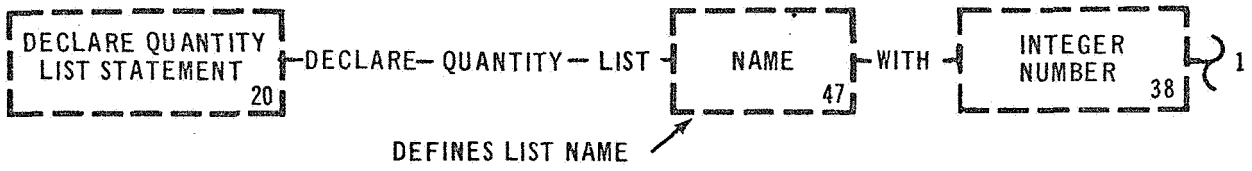
The DECLARE NUMERIC LIST STATEMENT is a Language Processor directive and is used to assign an arbitrary name and numeric data characteristics to a list of data. An unsigned Integer is required to indicate the number of variables in the list. A bypass is provided if no entries are present. A numeric variable may be declared to establish an initial value of a variable by setting it to a value defined by Number, Binary Number, Hexadecimal Number, Integer Number, or Octal Number.

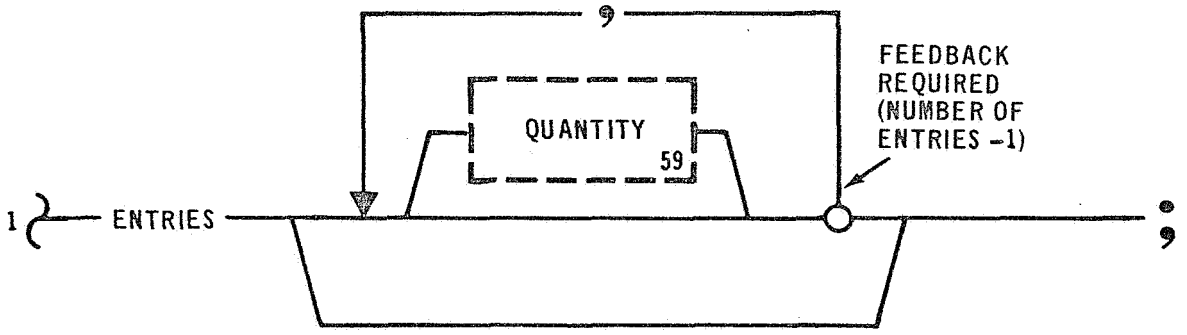
Examples: DECLARE NUMERIC LIST (OCTAL LIST) WITH 8 ENTRIES 3, 49,
X 3B, T9, B101, 64, 3.8, .6;

DECLARE NUMERIC LIST (LIST 1) WITH 12 ENTRIES
,3997,,,421,,,822,1,,10,,12;

20 REV 0

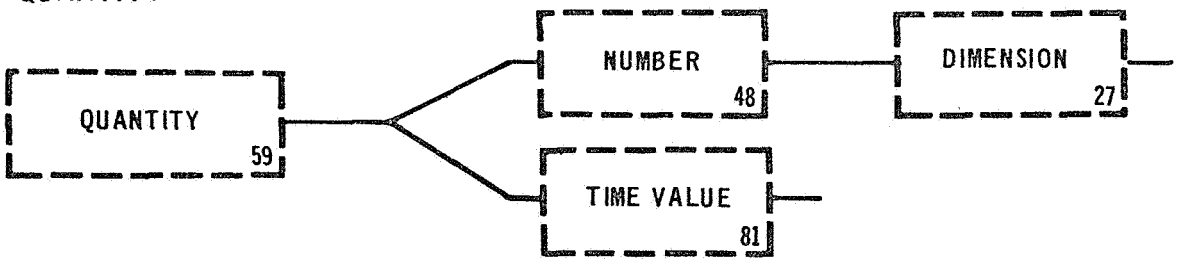
## DECLARE QUANTITY LIST STATEMENT




59 REV 0

## QUANTITY



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

## 2.2.2 Declare Quantity List Statement

Examples: DECLARE QUANTITY LIST (LIST A) WITH 3 ENTRIES;

DECLARE QUANTITY LIST (VOLTAGE LIST) WITH 6 ENTRIES
28V, +0.5V -0.5V, 0V, 50V, 10 SECS;

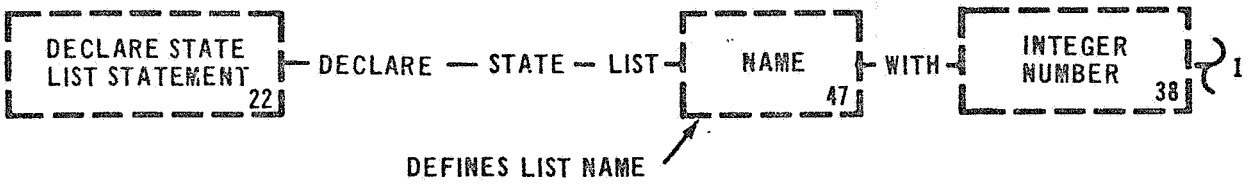
The DECLARE QUANTITY LIST STATEMENT is a Language Processor directive and is used to assign an arbitrary name to a list of variables of similar data characteristics. A Quantity is a Number followed by a Dimension. An unsigned Integer is required to indicate the number of variables in the list. A bypass is provided if no entries are present. A Quantity variable may be declared to establish an initial value by setting it to a decimal value followed by a Dimension identifier; (see page 200 for a list of legal dimensions).

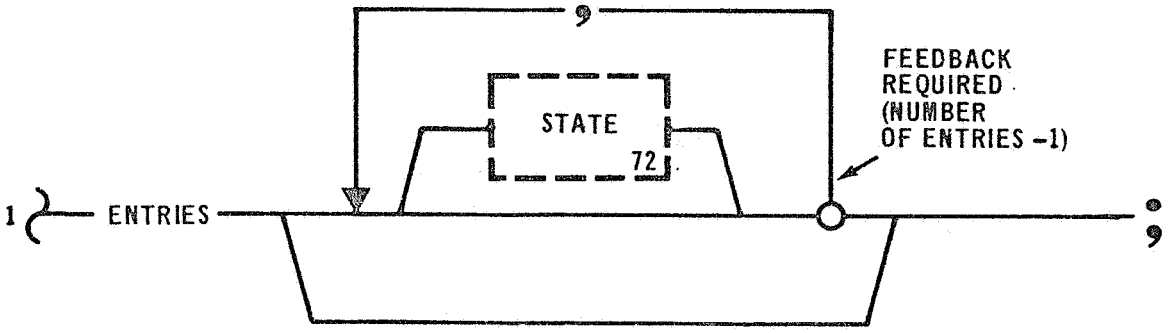
DECLARE QUANTITY LIST (ANALOG LIST) WITH 10 ENTRIES 200 PSIA,,,,,
6V,,20SECS,2HRS,212DEGF;

DECLARE QUANTITY LIST (QUANTITY LIST) WITH 9 ENTRIES 24V, 10A,
113 PSI, 10PCT, 2000REV, 2400FT/SEC, 2354KT,,;

22 REV 0

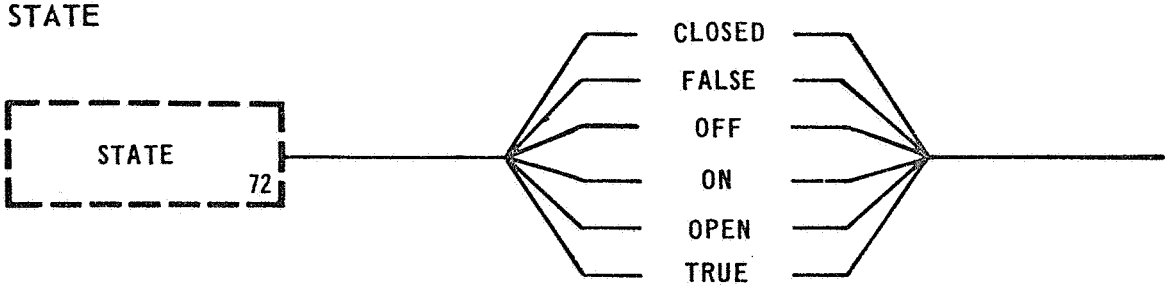
## DECLARE STATE LIST STATEMENT




72 REV 1

## STATE



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 2.2.3 Declare State List Statement

Examples: DECLARE STATE LIST (FLAG LIST) WITH 10 ENTRIES;

DECLARE STATE LIST (LIST STATE) WITH 6 ENTRIES

ON, ON, ON, OFF, OFF, ON;

The DECLARE STATE LIST STATEMENT is a Language Processor directive and is used to name a list of variables of similar data characteristics. An unsigned Integer is required to indicate the number of variables in a list. A bypass is provided if no entries are present. A State variable may be declared for use as an internal program flag and may be set to an initial value.

Examples: DECLARE STATE LIST (LIST A) WITH 6 ENTRIES

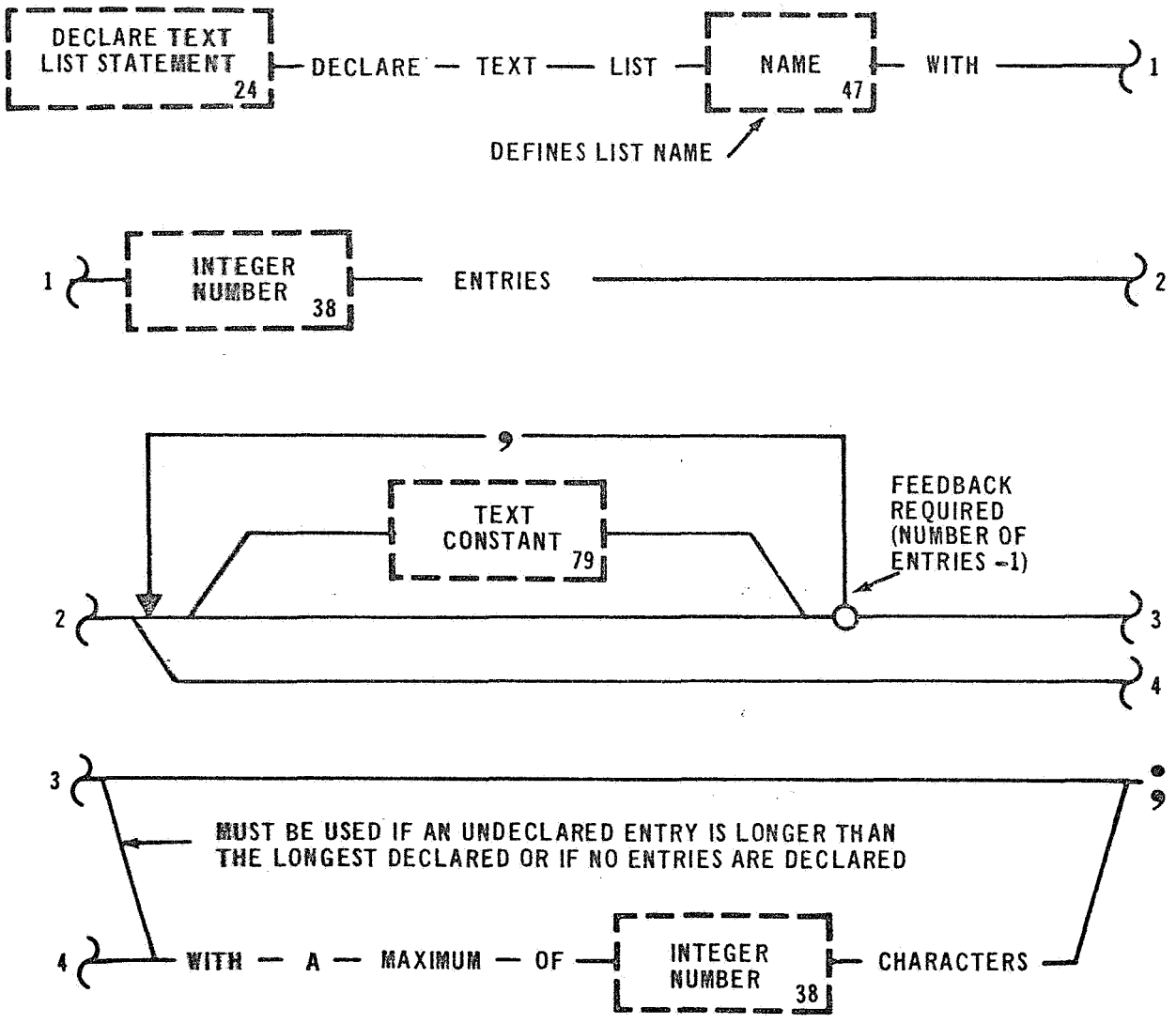
TRUE, OPEN, CLOSED, FALSE, ON, OFF;

DECLARE STATE LIST (LIST 3) WITH 3 ENTRIES

OPEN, OPEN, OPEN;

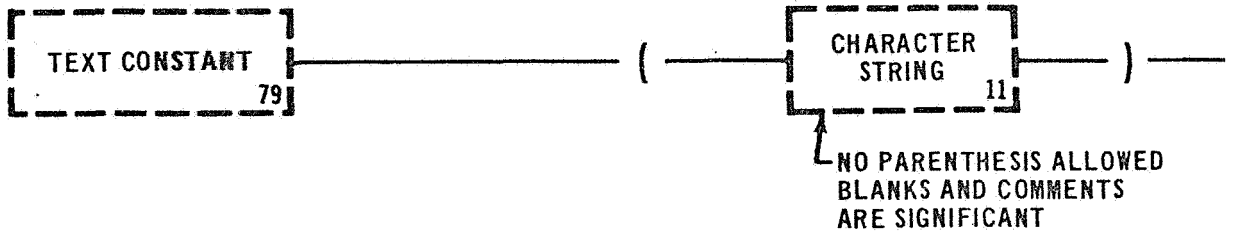
24 REV 1

## DECLARE TEXT LIST STATEMENT




79 REV 0

## TEXT CONSTANT



| DECLARATION |
|---|
| PROCEDURAL |
| SYSTEM |

## 2.2.4 Declare Text List Statement

Examples: DECLARE TEXT LIST (INPUT) WITH 2 ENTRIES WITH A
MAXIMUM OF 25 CHARACTERS;
DECLARE TEXT LIST (OPERATOR INSTRUCTION) WITH 2
ENTRIES (PLACE SWITCHES INDICATED),
(*PREFLIGHT TM CAL IN PROGRESS *);

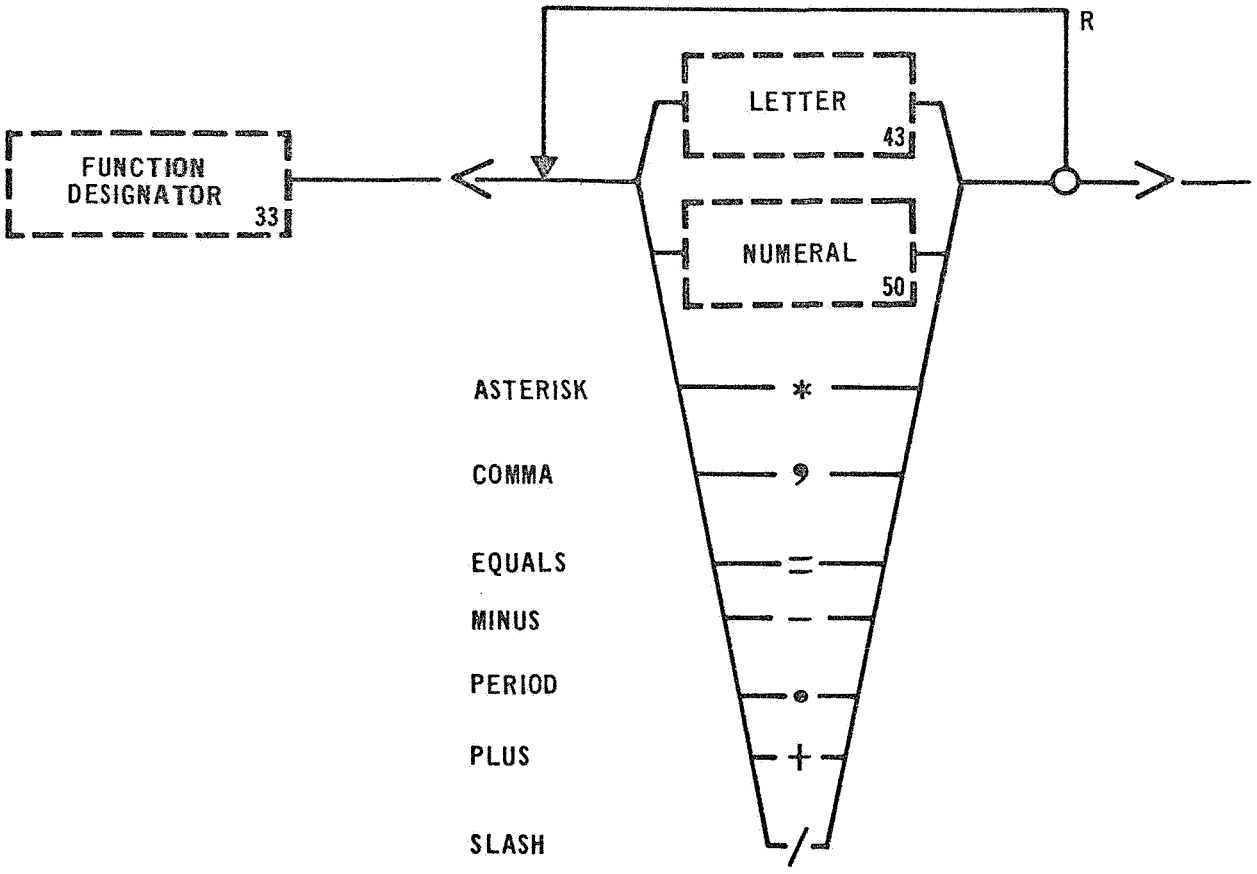
The DECLARE TEXT LIST STATEMENT is a Language Processor directive and is used to name a list of Text data. An unsigned Integer is required to indicate the number of variables in the list. A bypass is provided if no entries are present. A Text variable may be declared for use as a program message or may be used to reserve storage for input messages during program execution. If a Name is used to reserve storage, the number of characters is expressed as an unsigned Integer. The unsigned Integer is required only if an undeclared entry is longer then the longest declared message.

Examples: DECLARE TEXT LIST (DISPLAY MESSAGE) WITH 5 ENTRIES
(*PREFLIGHT TM CAL IN PROGRESS*),
(*PREFLIGHT 00 LEVEL IN PROGRESS*),
(*PREFLIGHT 25 LEVEL IN PROGRESS*),
(*PREFLIGHT 50 LEVEL IN PROGRESS*),
(*PREFLIGHT 100 LEVEL IN PROGRESS*);

```
DECLARE TEXT LIST (INPUT OUTPUT MESS) WITH 6 ENTRIES , , , ,
(PLACE ABOVE SWITCHES AS INDICATED), (SWITCH SCAN IN
PROGRESS) WITH A MAXIMUM OF 36 CHARACTERS;
```

33
REV 0

## FUNCTION DESIGNATOR



## 2.3 DECLARE TABLES

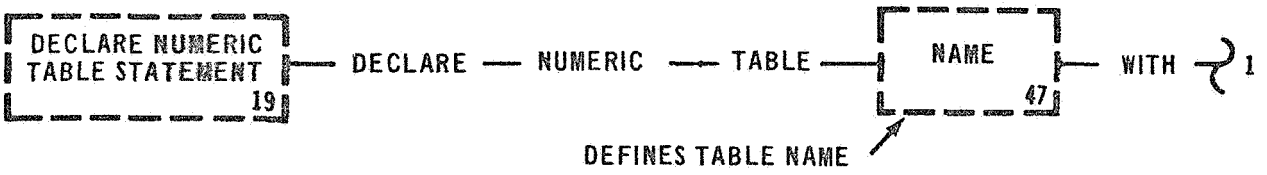
A table is used when a group of Function Designators is, collectively, assigned a name. Function Designators are system names and are made available in the Language Processor by the Data Bank. A table can be visualized in the conventional row/column format with Function Designators framing the rows and with Column Names heading the columns. These items are fixed items and may not be manipulated during execution. Data is entered via a Declaration Statement a row at a time. There is a comma required to signal the end of the row. This assures unique definition of a table without data columns and a table of one data column where the data is not specified. A Table Name must be unique, that is, a Table Name can reference only one table within a GOAL component. Column Names must also be unique within a GOAL component.

| | TABLE NAME | | | | |
|---|---|---|---|---|---|
| | | 1 COLUMN NAME | 2 COLUMN NAME | 3 COLUMN NAME | Cn COLUMN NAME |
| Row 1 | 1 FUNCTION DESIGNATOR | data 11 | data 12 | data 13 | data 1N |
| Row 2 | 2 FUNCTION DESIGNATOR | data 21 | data 22 | data 23 | data 2N |
| Row 3 | 3 FUNCTION DESIGNATOR | data 31 | data 32 | data 33 | data 3N |
| Row Rn | Rn FUNCTION DESIGNATOR | data R1 | data R2 | data R3 | data RN |
| | | 1 | 2 | 3 | Cn |

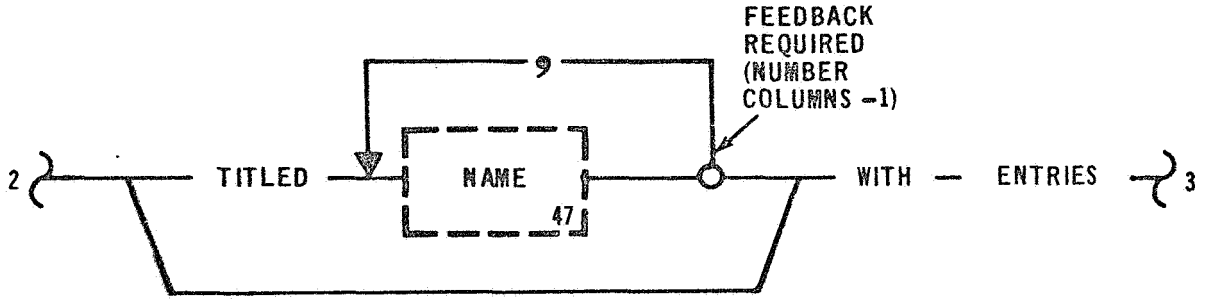
Column Numbers

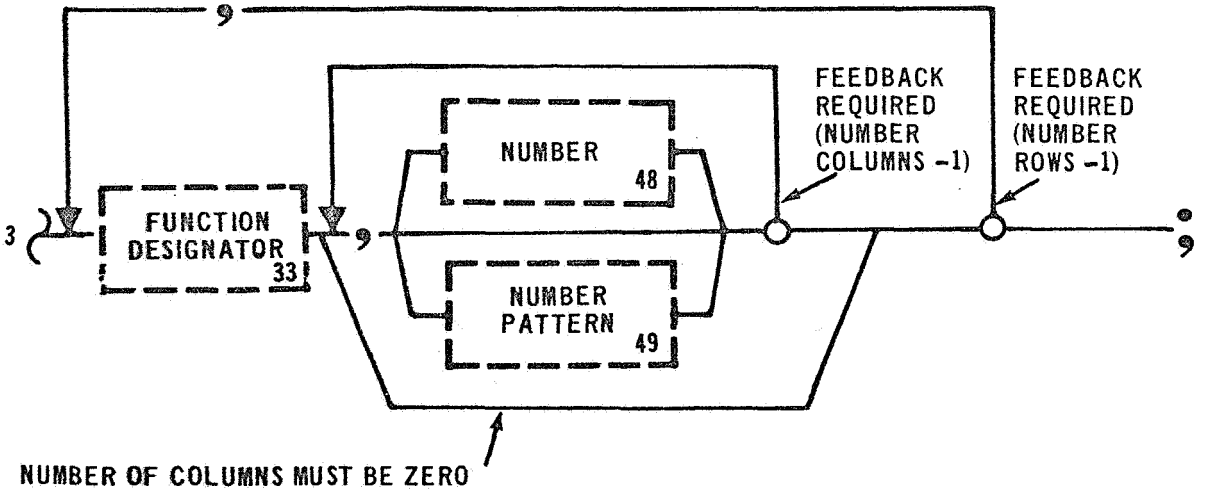
DECLARE NUMERIC TABLE STATEMENT

# DECLARE NUMERIC TABLE







| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

## 2.3.1 Declare Numeric Table Statement

Examples: DECLARE NUMERIC TABLE (HIGH LOW RUN) WITH 3 ROWS AND
4 COLUMNS TITLED
(HIGH), (LOW), (RUN), (CUR) WITH ENTRIES
<E1 GG CHAMBER P> , 1000.1, 1.0, 500.0, ,
<E2 GG CHAMBER P> , 1001.2, .9, 500.0, ,
<E3 GG CHAMBER P> , 999.8, 1.2, 500.0, ;

The DECLARE NUMERIC TABLE STATEMENT is used to assign an arbitrary name to a numeric table containing information pertaining to Specified Function Designators. The Table Name must be unique within a GOAL component. A Table Name is an internal variable and is used as an indirect linkage for the Function Designators included in the table. For example: A READ STATEMENT that refers to a specific Table Name will read the test conditions of the activated Function Designators contained within the table, and not the contents of the table.

Unsigned Integers are required to indicate the number of rows and columns contained within a table. The Language Processor will verify that the indicated numbers equal the number of rows and columns used.

Columns may or may not be titled, but if one column is assigned a name all other columns within the table must be titled. Column Names must be unique. Columns within other tables in the same GOAL component

cannot have the same Column Name.

A Function Designator is required to start each row and may be used to identify that row.

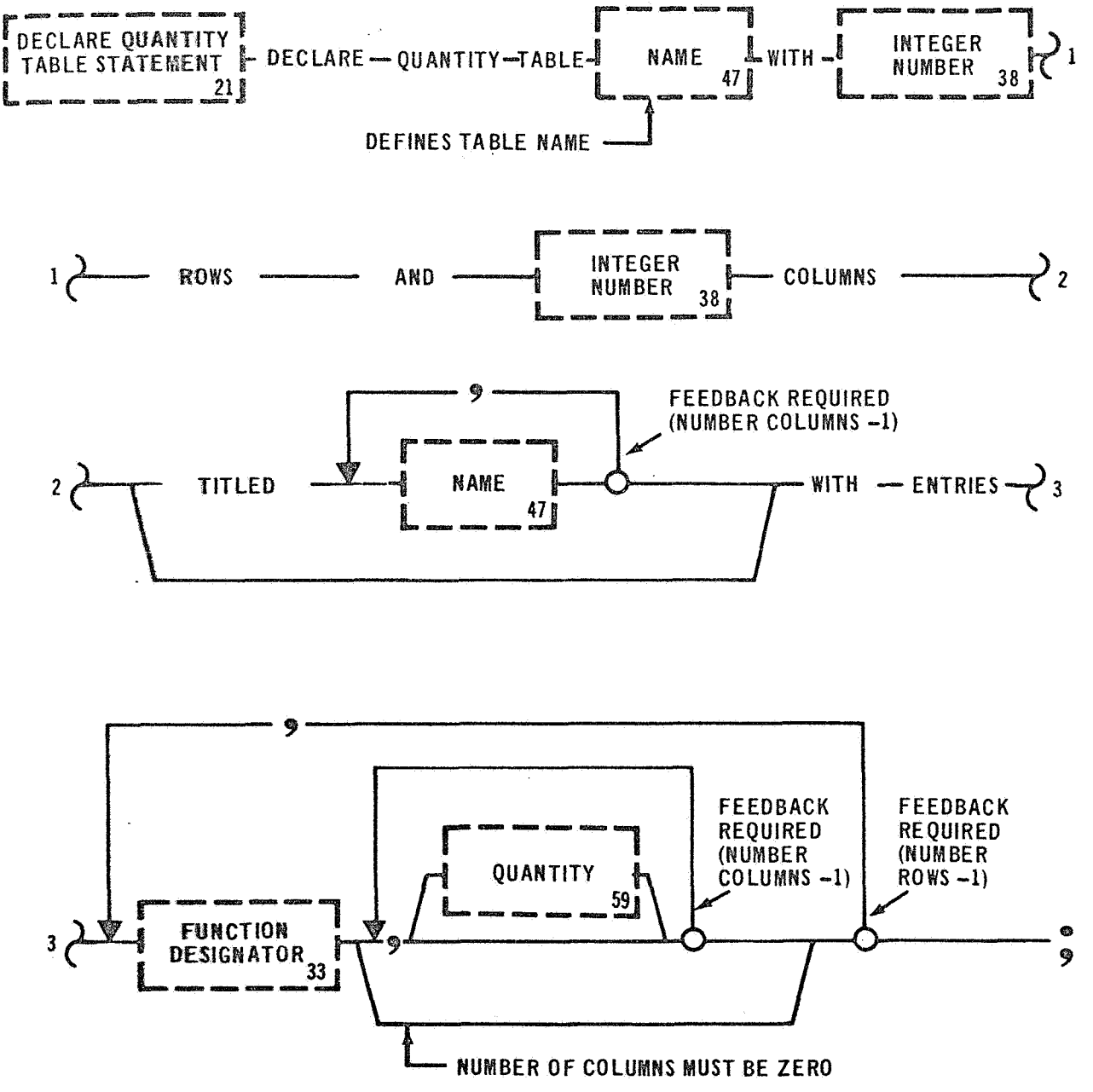
Data locations within a table may be set to an initial value as described by the Number or Number Pattern syntax diagrams. Data locations may also be used to reserve storage which is to be used during program execution to save data.

Examples: DECLARE NUMERIC TABLE (LIMIT TABLE) WITH 5 ROWS AND 2 COLUMNS TITLED

| | (COL 1), | (COL 2) WITH ENTRIES |
|---|---|---|
| <E1 LOX TURBINE P>, | 2000.1, | 0.4, |
| <E2 LOX TURBINE P>, | 2010.2, | 0.2, |
| <E3 LOX TURBINE P>, | 2002.4, | 0.6, |
| <E4 LOX TURBINE P>, | 2003.1, | 0.2, |
| <E5 LOX TURBINE P>, | 2011.9, | -0.1; |

21 REV 1

## DECLARE QUANTITY TABLE STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

## 2.3.2 Declare Quantity Table Statement

Examples: DECLARE QUANTITY TABLE (MAIN FUEL FLOW) WITH 5 ROWS
AND 3 COLUMNS WITH ENTRIES
< E1 MAIN FUEL > , 0.1 PPS, 300.1 PPS, ,
< E2 MAIN FUEL > , 0.3 PPS, 300.2 PPS, ,
< E3 MAIN FUEL > , 0.4 PPS, 300.1 PPS, ,
< E4 MAIN FUEL > , 0.2 PPS, 300.1 PPS, ,
< E5 MAIN FUEL > , 0.1 PPS, 299.8 PPS, ;

DECLARE QUANTITY TABLE (TURBINE DATA TABLE) WITH 2 ROWS
AND 4 COLUMNS TITLED (TURBINE OUTLET PRESSURE),
(TURBINE INLET PRESSURE), (TURBINE OUTLET TEMP),
(TURBINE INLET TEMP), WITH ENTRIES < LOX TURBINE
NO 1 > , , , , , < LOX TURBINE NO 2 > , , , , ;

The DECLARE QUANTITY TABLE STATEMENT is used to assign an arbitrary name to a table of variables with similar data characterisitcs. A Table Name must be unique as are all other names within a test program. An unsigned Integer is required to declare the number of rows within a table. A second unsigned Integer is necessary to declare the number of columns within a table. The Language Processor will insure that the number or rows and columns declared equal the number of rows and columns used.

Data positions within a table may be set to an initial value as

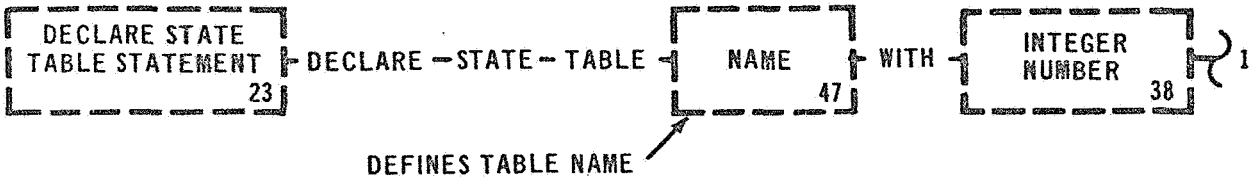
described by Quantity syntax diagram.

Storage may be declared for use during program execution by taking the bypass around the data entry.

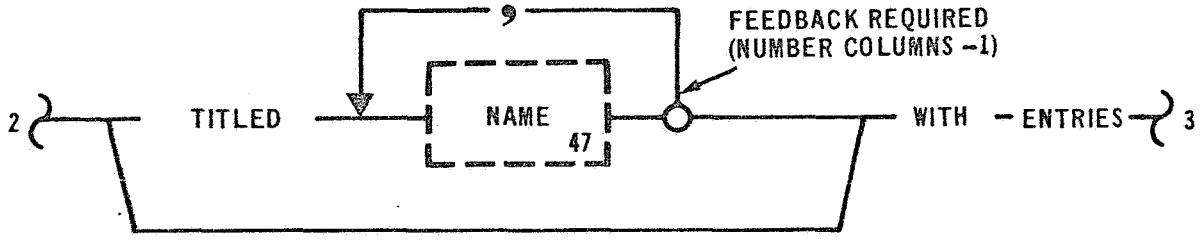
Examples: DECLARE QUANTITY TABLE (EDS SWITCH POINT) WITH 3 ROWS AND
2 COLUMNS TITLED
(HIGH), (LOW) WITH ENTRIES
<EDS SWITCH POINT 1> , 28.0V, 0.0,
<EDS SWITCH POINT 2> , 28.1V, 0.1,
<EDS SWITCH POINT 3> , 28.3V, 0.2;
DECLARE QUANTITY TABLE (TIME) WITH 2 ROWS AND 2 COLUMNS TITLED
(CLOCK), (GMT) WITH ENTRIES
<CLOCK1> , -2 HRS 30 MINS, 10 HRS 30 MINS,
<CLOCK2> , -1HR 30 MINS, 12 HRS 30 MINS;

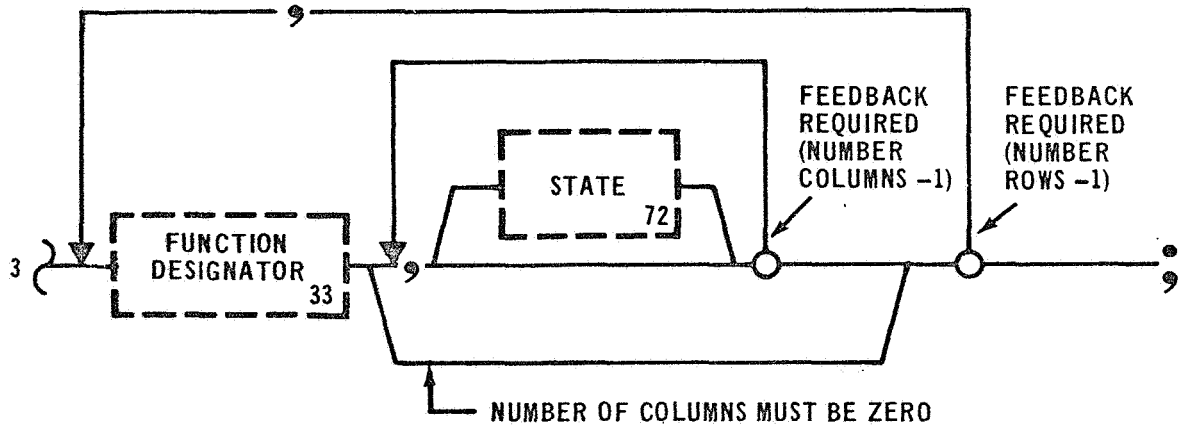
23
REV 0

## DECLARE STATE TABLE STATEMENT







| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

## 2.3.3 Declare State Table Statement

Examples: DECLARE STATE TABLE (THRUST OK) WITH 5 ROWS AND 3 COLUMNS
TITLED (THRUST OK), (THRUST NOT OK), (STATE) WITH ENTRIES
< THRUST OK 1E1 > , ON, OFF, ,
< THRUST OK 1E2 > , ON, OFF, ,
< THRUST OK 1E3 > , ON, OFF, ,
< THRUST OK 1E4 > , ON, OFF, ,
< THRUST OK 1E5 > , ON, OFF, ;

The DECLARE STATE TABLE STATEMENT is used to assign an arbitrary name to a table of variables with State data characteristics. A Table Name must be unique. A Table Name is an internal variable and may be used in an External Test Action Statement (see Procedural Statement) to reference test points on the system under test. For example: A Set Statement that refers to a Table Name will result in the activated Function Designators listed in that table being issued to the system under test in the indicated state.

An unsigned Integer is required to declare the number of rows, an additional unsigned Integer is required to declare the number of columns.

Columns may or may not be titled, but if a column is assigned a name all other columns within the table must be titled. Column names must

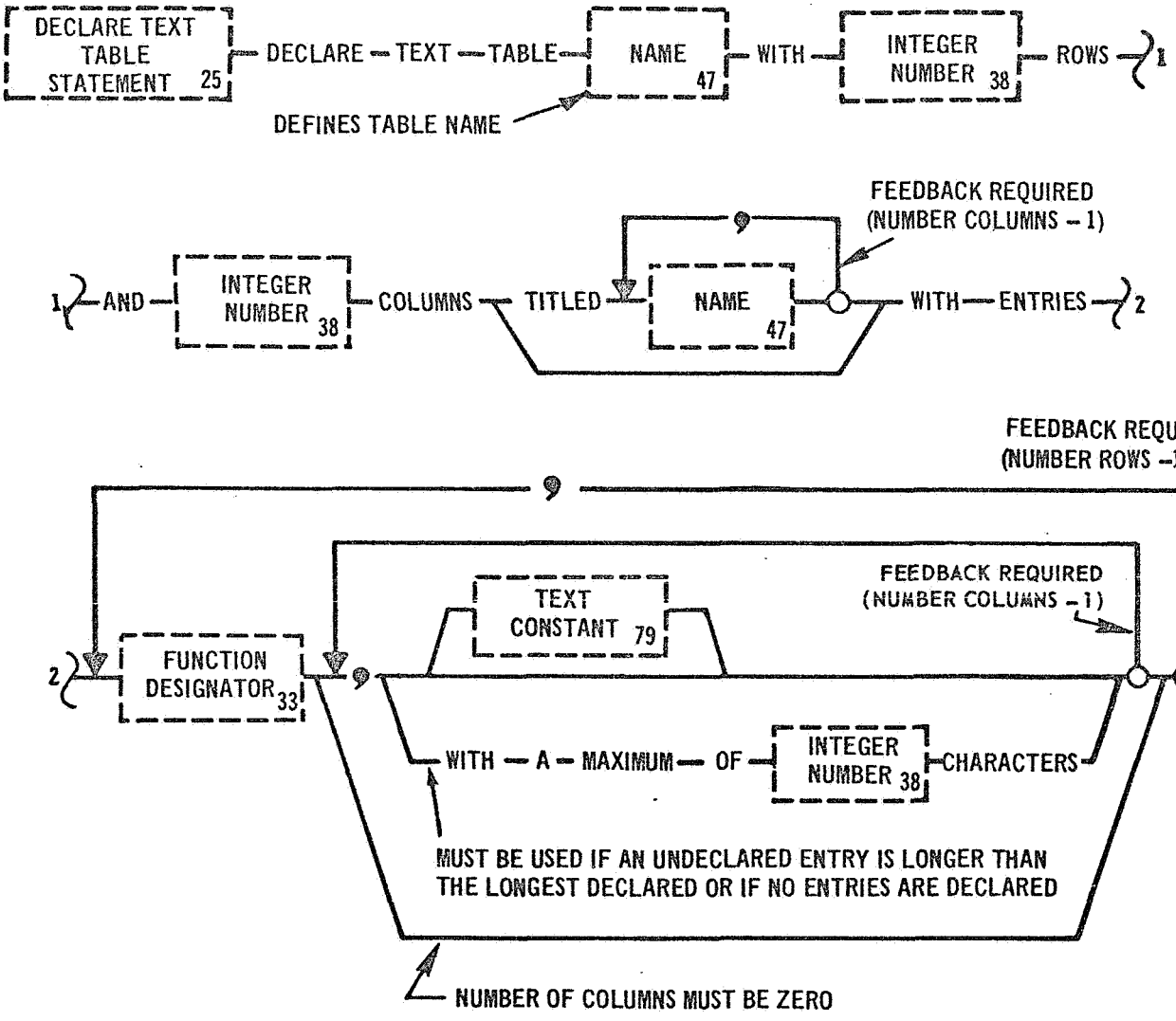
be unique.

The data positions of a table may be set to an initial state. The data positions may have data characteristics as described by OPEN, CLOSED, TRUE, FALSE, ON, or OFF.

Examples: DECLARE STATE TABLE (CONTROL SOLENOID TABLE) with 3 ROWS
AND 2 COLUMNS WITH ENTRIES
<E1 HE CONTROL SOLENOID> , , ,
<E2 HE CONTROL SOLENOID> , , ,
<E3 HE CONTROL SOLENOID> , , ;
DECLARE STATE TABLE (LIMIT SWITCH) WITH 10 ROWS AND 4
COLUMNS WITH ENTRIES
<LIMIT SWITCH FUEL NO 1E1> , ON, OFF, ON, OFF,
<LIMIT SWITCH FUEL NO 2E1> , OFF, ON, ON, OFF,
<LIMIT SWITCH FUEL NO 1E2> , ON, OFF, ON, OFF,
<LIMIT SWITCH FUEL NO 2E2> , OFF, ON, ON, OFF,
<LIMIT SWITCH FUEL NO 1E3> , ON, OFF, ON, OFF,
<LIMIT SWITCH FUEL NO 2E3> , OFF, ON, ON, OFF,
<LIMIT SWITCH FUEL NO 1E4> , ON, OFF, ON, OFF,
<LIMIT SWITCH FUEL NO 2E4> , OFF, ON, ON, OFF,
<LIMIT SWITCH FUEL NO 1E5> , ON, OFF, ON, OFF,
<LIMIT SWITCH FUEL NO 2E5> , OFF, ON, ON, OFF;

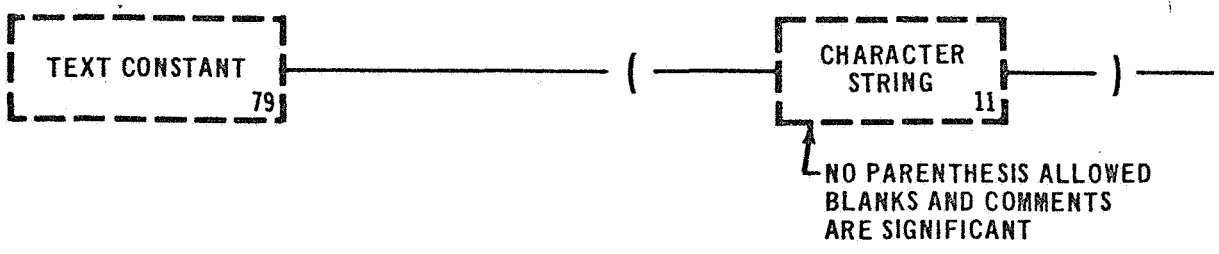
25 REV 1

## DECLARE TEXT TABLE STATEMENT




79 REV 0

## TEXT CONSTANT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

## 2.3.4 Declare Text Table Statement

Examples: DECLARE TEXT TABLE (MESSAGE TABLE) WITH 2 ROWS AND 1 COLUMN
TITLED
(MESSAGE A) WITH ENTRIES
< 224 DISPLAY B35 > , (SWITCH SCAN IN PROGRESS),
< 224 DISPLAY B42 > , (PLACE ABOVE SWITCHES AS INDICATED);

The DECLARE TEXT TABLE STATEMENT is used to assign an arbitrary name to a table of messages. A Table Name must be unique.

An unsigned Integer is used to declare the number of rows, another unsigned Integer is used to declare the number of columns. The Language Processor will verify that the number of rows and columns declared equal the number used in the table.

Columns may be assigned an arbitrary name, but if one column is titled all columns in the table must be titled. Column Names are used as an identifier for a specific column, and must be unique.

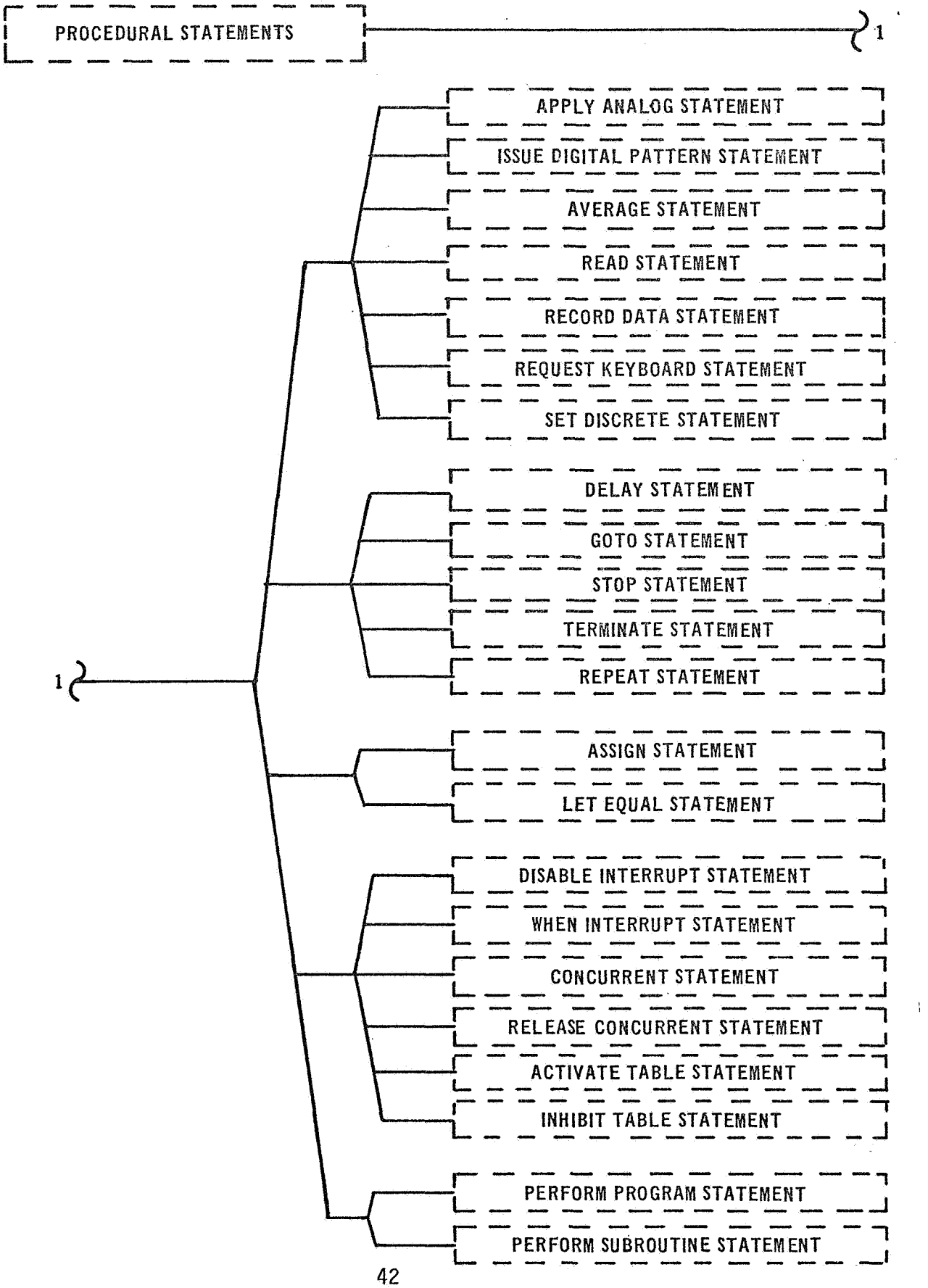
A Function Designator is required to start each row and is used as an identifier for a specific row.

A Text Constant may be declared and used as a precanned message.

Storage may also be reserved for execution time input, by taking the bypass around the Text Constant entry. If the number of characters in

an undeclared entry is greater than the number of characters in a declared message, the maximum number of characters in the longest undeclared message must be entered as an unsigned Integer. Also, if no entries are present in the table, the maximum number of characters must be indicated.

Examples: DECLARE TEXT TABLE (INPUT TABLE) WITH 2 ROWS AND 1 COLUMN
TITLED
(INPUT MESSAGE) WITH ENTRIES
<224 DISPLAY A21> ; ,
<224 DISPLAY A28> , WITH A MAXIMUM OF 32 CHARACTERS;
DECLARE TEXT TABLE (MESSAGE TABLE NO. 1) WITH 4 ROWS AND
2 COLUMNS TITLED
(MESS 1), (MESS 2) WITH ENTRIES
<224 MESS A32> , , ,
<224 MESS A31> , , ,
<224 MESS A32> , , ,
<224 MESS A33> , ,
WITH A MAXIMUM OF 10 CHARACTERS;


PROCEDURAL STATEMENTS
1
1
APPLY ANALOG STATEMENT
ISSUE DIGITAL PATTERN STATEMENT
AVERAGE STATEMENT
READ STATEMENT
RECORD DATA STATEMENT
REQUEST KEYBOARD STATEMENT
SET DISCRETE STATEMENT
DELAY STATEMENT
GOTO STATEMENT
STOP STATEMENT
TERMINATE STATEMENT
REPEAT STATEMENT
ASSIGN STATEMENT
LET EQUAL STATEMENT
DISABLE INTERRUPT STATEMENT
WHEN INTERRUPT STATEMENT
CONCURRENT STATEMENT
RELEASE CONCURRENT STATEMENT
ACTIVATE TABLE STATEMENT
INHIBIT TABLE STATEMENT
PERFORM PROGRAM STATEMENT
PERFORM SUBROUTINE STATEMENT

# SECTION III, PROCEDURAL STATEMENTS

## 3.0 GENERAL

Procedural Statements are executable statements. These statements control the testing sequence by issuing commands, acquiring data from the system under test, controlling the internal program execution sequence, and manipulating data.

The Procedural Statements are functionally grouped and are presented in the following order:

EXTERNAL TEST ACTION

INTERNAL SEQUENCE CONTROL

ARITHMETIC/LOGICAL OPERATIONS

EXECUTION CONTROL

INTERRUPT CONTROL

TABLE CONTROL

## 3.1 PROCEDURAL STATEMENTS PREFIX

The Procedural Statements are the only statements that may have a statement prefix. It is this prefix that allows a statement to have a Statement Number and to be executed based on a variety of conditions.

Because the statement prefix is optional for any Procedural Statement, the composition and capabilities of the statement prefix are discussed prior to the individual Procedural Statement. The Procedural Statement Prefix has three optional capabilities: the Statement Number, Time Prefix, and the Verify Prefix.

The Statement Number, if used, must be the first entry in a statement. It precedes the Time Prefix and the Verify Prefix. Statement Numbers are used to identify a specific Procedural Statement, and therefore,

must be unique. The Language Processor will flag as an error any multiple defined Statement Numbers. However, a Statement Number may be referenced as many times as necessary. By convention, it is recommended that Statement Numbers be used in ascending order. This will enhance procedure review.

The Time Prefix cannot precede a Statement Number. The Time Prefix provides for synchronization of execution of a Procedural Statement with an external clock. Synchronization is achieved by delaying execution until a specific Time Value is reached or passed. The remaining portion of the statement may contain a Verify Prefix.

The Verify Prefix, if used, cannot precede a Statement Number or Time Prefix. The Verify Prefix provides a conditional transfer capability depending upon the results of a comparison type test.

### 3.1.1 External Test Action

The External Test Action Statements are those statements that provide an interface with the system under test. This group of statements also includes the necessary statements for communication between test engineer/computer system, computer system/computer system, and test system/computer system. The External Test Action Statements are subdivided into Command and Response categories.

### 3.1.2 Command Statements

The Command Statements are those statements that stimulate the system under test. The test points on the system under test are defined as Function Designators. Function Designators are items which interface via the Data Bank with the system under test. A test point receiving

a command issued by this subgroup must be specified as a load type Function Designator. A Function Designator used in the RECORD DATA STATEMENT may be either a load or system type, depending upon the individual test point. The External Designator syntax diagram allows referencing of either a single Function Designator or a group of Function Designators.

The Command Statements consist of the following statements:

APPLY ANALOG STATEMENT

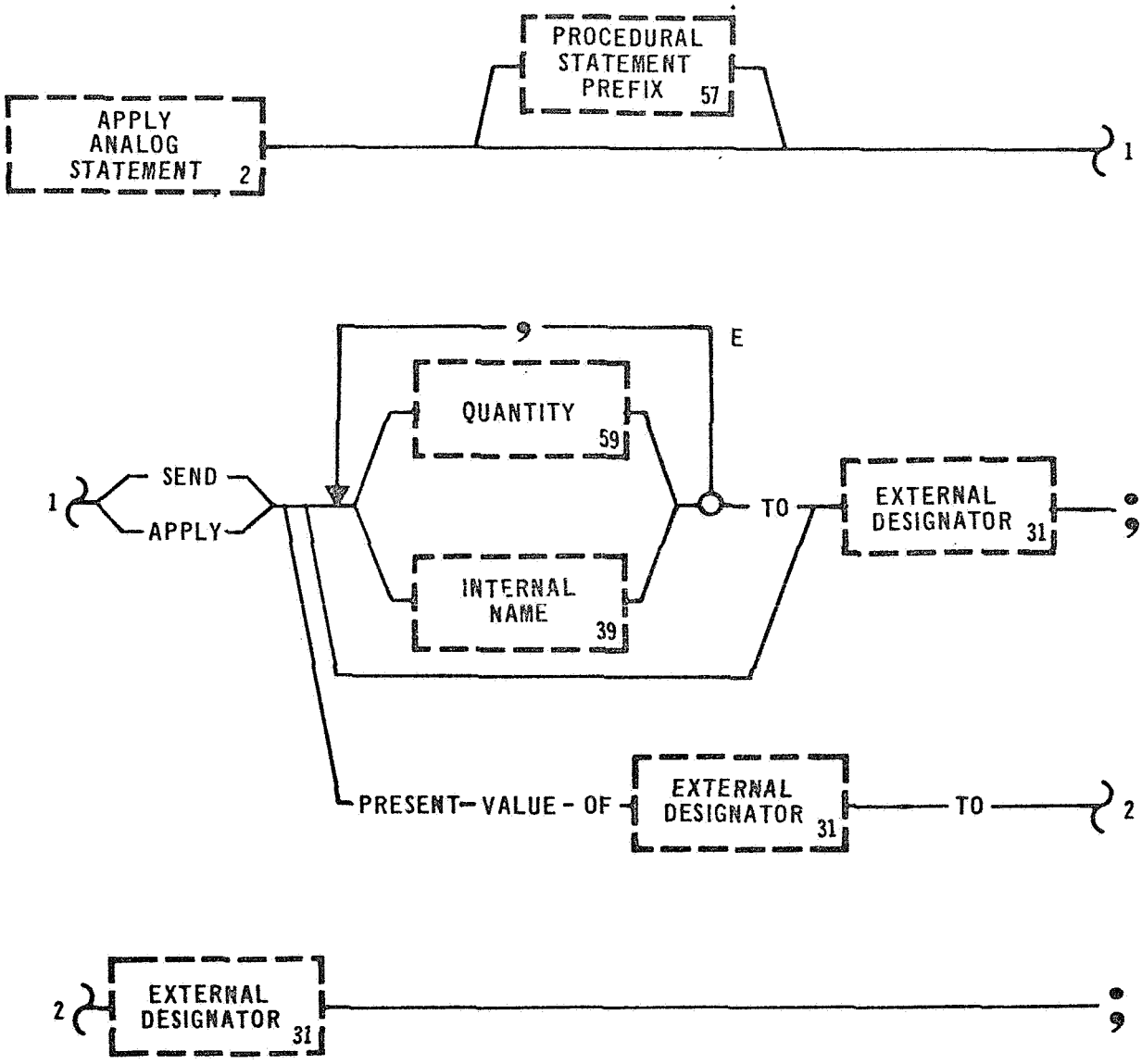
ISSUE DIGITAL PATTERN STATEMENT

SET DISCRETE STATEMENT

RECORD DATA STATEMENT

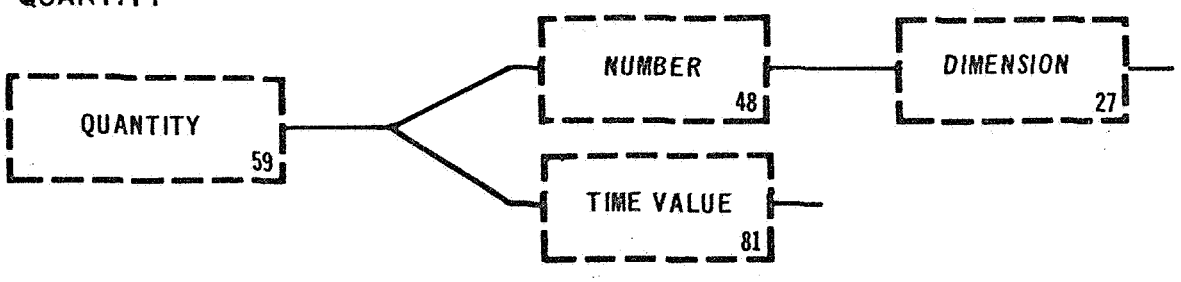
2 REV 0

## APPLY ANALOG STATEMENT




59 REV 0

## QUANTITY



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

3.1.2.1 Apply Analog Statement

Examples: APPLY 10V TO <RELAY NO 101> ;

SEND (POWER) TO <HYDRAULIC SYSTEM> ;

The APPLY ANALOG STATEMENT provides for an analog stimuli to be sent to a system under test. A Quantity or an Internal Name identifying a Quantity may be sent to the system under test by this statement. The Language Processor will verify that the data referenced in this statement was previously declared to be a Numeric or Quantity type. One Quantity may be specified, and it may be sent to one or more Function Designators, or several Quantities may be specified and sent to several Function Designators. In this option the number of Quantities must equal the number of Function Designators in the statement, or in referenced tables. The quantities will be applied, in the order in which they appear in the statement, to the Function Designators as they appear in the same statement, or in referenced tables. The Language Processor will verify a one-to-one relationship of Function Designators to the Quantities to be applied. It will also verify that the listed Function Designators in this option are load type Data Bank resident items. When this statement is used to set the present value of one device to another, the processor will verify the devices are analog type devices and then generate an equivalent "READ" from the first Function Designator followed by a "SEND" to the

corresponding Function Designator. The first Function Designator(s) in this option must be specified in the Data Bank as sensor types. The second Function Designator(s) must be specified as load types.

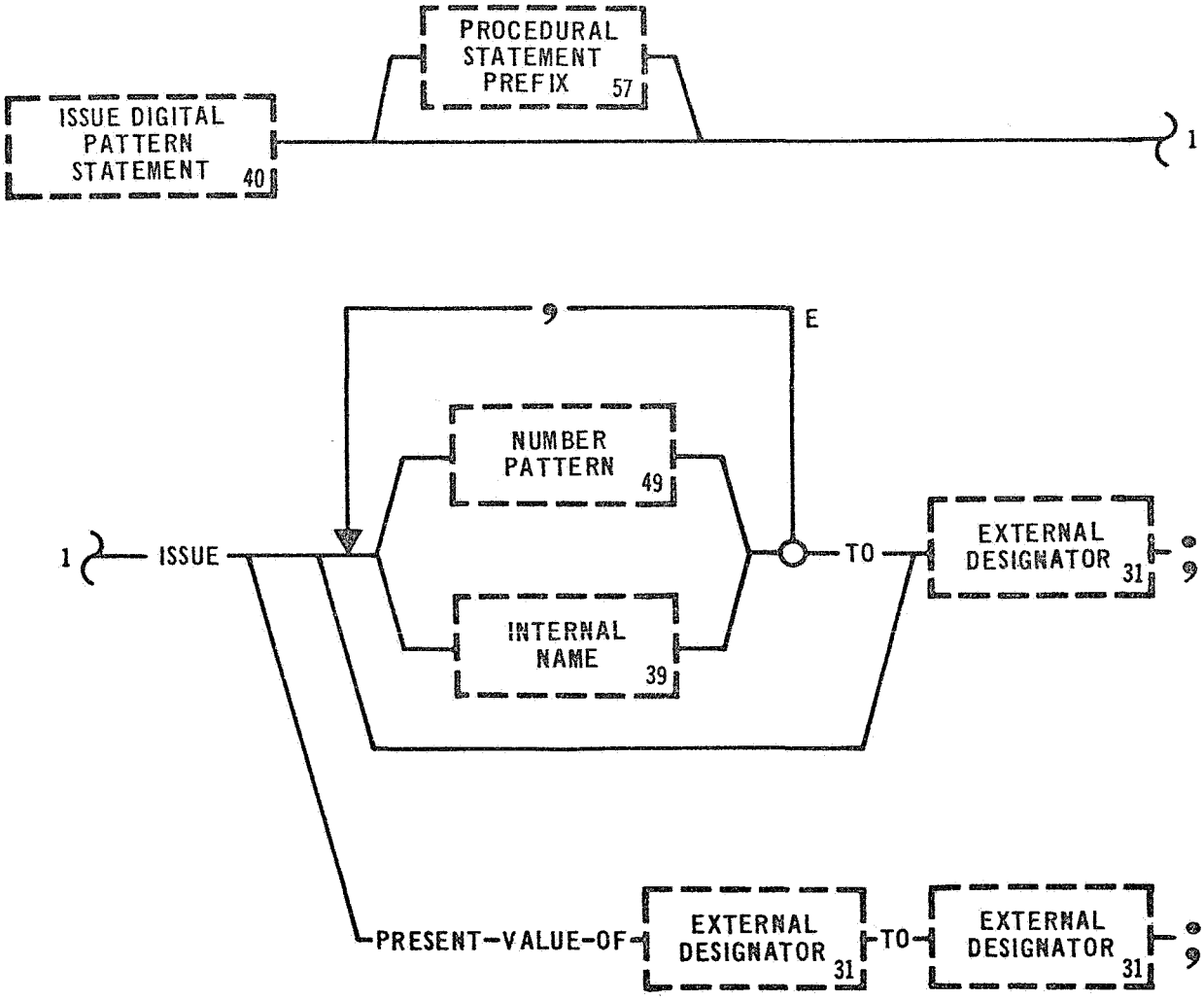
Examples: SEND 10V TO <POWER SELECTOR 1>,<POWER SELECTOR 2>;

APPLY 10V, 5V, 15V TO <POWER BUS 1>,<POWER BUS 2>,
<POWER BUS 3> ;

APPLY PRESENT VALUE OF <POWER BUS 1> TO <POWER BUS 2>;

40 REV 0

## ISSUE DIGITAL PATTERN STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

3.1.2.2 Issue Digital Pattern Statement

Examples: ISSUE <S4B SWITCH SELECTOR CH 59> ;

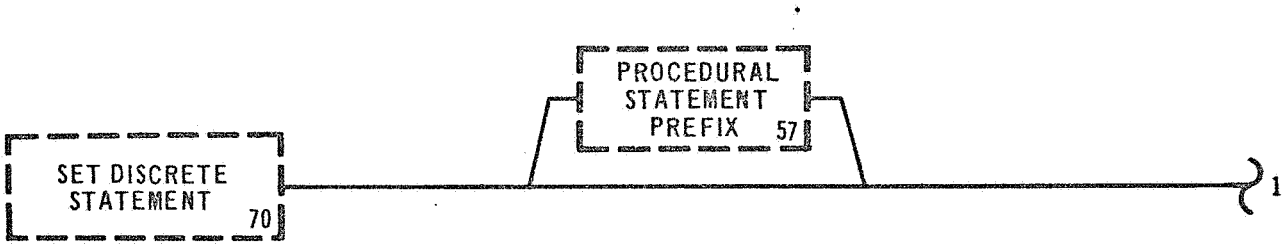
ISSUE B 111 110 101 100 011 TO <PANEL AB> ;

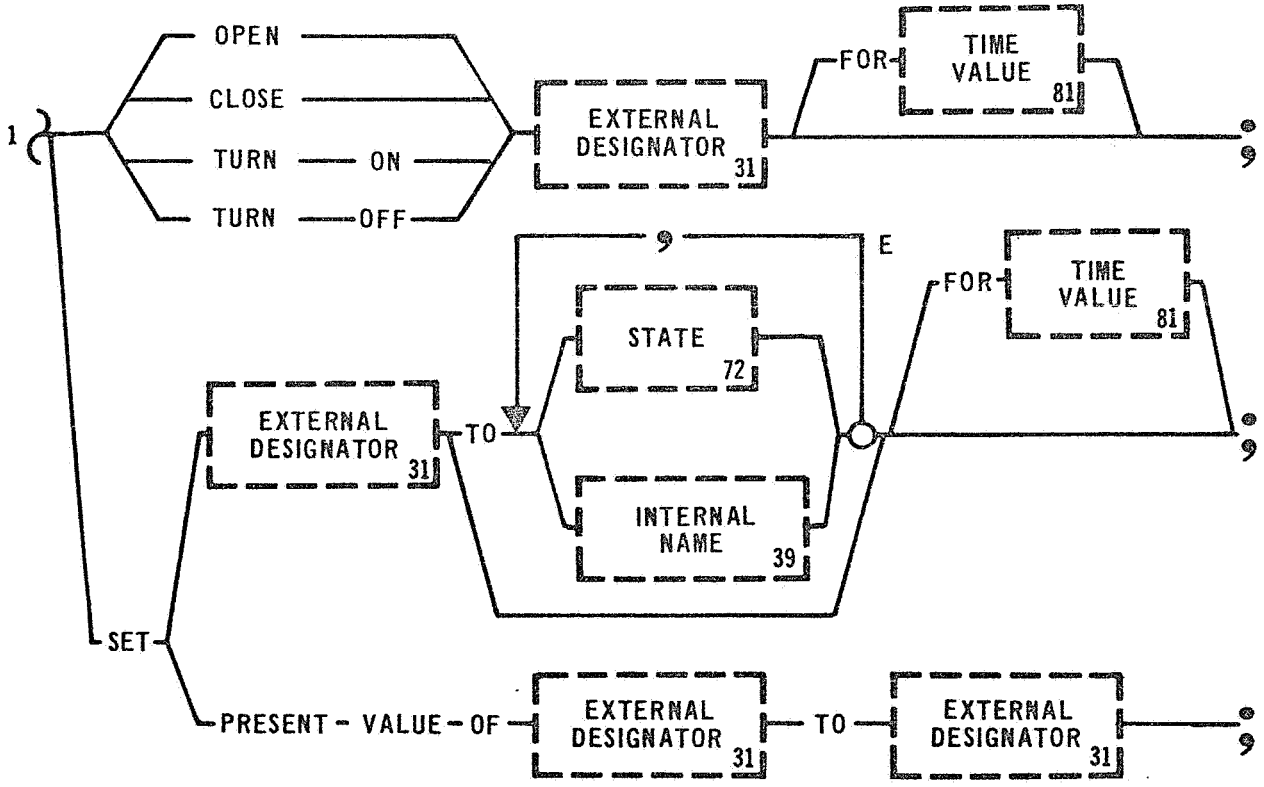
The ISSUE DIGITAL PATTERN STATEMENT issues a digital pattern to a system under test. A Number Pattern or an Internal Name declared as a Number Pattern may be issued to the system under test. The Language Processor will verify that the referenced Internal Name was previously declared as numeric data. If more than one Number Pattern is to be issued to more than one Function Designator, the first Number Pattern will be issued to the first Function Designator, etc. In the case of a table the first Number Pattern specified will be issued to the first Row Designator. This type of one-to-one correspondence will continue until the last Number Pattern is issued. An inactive row will not be issued. One Number Pattern may be issued to several Function Designators. When this statement is used to issue the present value of one device to another, the processor will verify the devices are capable of this type operation and then generate an equivalent "READ" from the first Function Designator(s) followed by an "ISSUE" to the corresponding Function Designator(s).

Examples: ISSUE (OCTAL SEVENS), (OCTAL ONES) TO
<PANEL LIGHTS 32>,<PANEL LIGHTS 31> ;
ISSUE PRESENT VALUE OF<CH 63> TO<CH 11> ;

70 REV 0

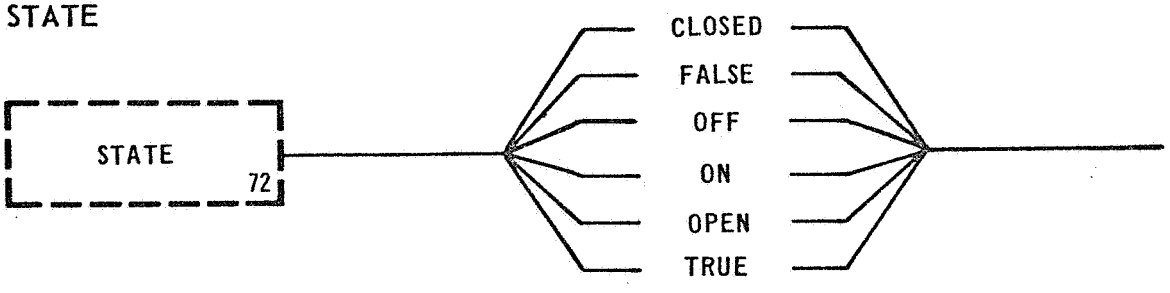
## SET DISCRETE STATEMENT




72 REV 1

## STATE



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

3.1.2.3 Set Discrete Statement

Examples: SET <POWER SELECTOR> TO (PRELAUNCH);

TURN ON <CUTOFF RESET> FOR 3 MSECS;

OPEN <LOX VENT VALVE 1> , <LOX VENT VALVE 2> ;

The SET DISCRETE STATEMENT allows discrete (ON/OFF) type States to be sent to the system under test. The Function Designator must be specified as a load discrete type command. This is the only Command Statement that allows a Time Value controlling the duration of the action. After the time delay has expired then the complement state is issued. The host component will then continue execution with the next sequential statement.

This statement can be used to control such items as panel indicators, equipment power, and other bi-state units, where a very small Time Value is required.

During automatic execution, this statement can also be used in setting system indicators (flags) to be used by another program at some subsequent point in testing.

In the Data Bank the Language Processor will verify that data referenced in this statement was previously declared to be State type.

This statement may be used to set a test clock to a specified Time

Value. Time devices are special and are specified accordingly in the Data Bank.

When the statement is used to set the present value of one device to another, the processor will verify the devices are discrete type devices and then generate an equivalent "READ" from the first Function Designator(s) followed by a "SET" to the corresponding Function Designator(s).

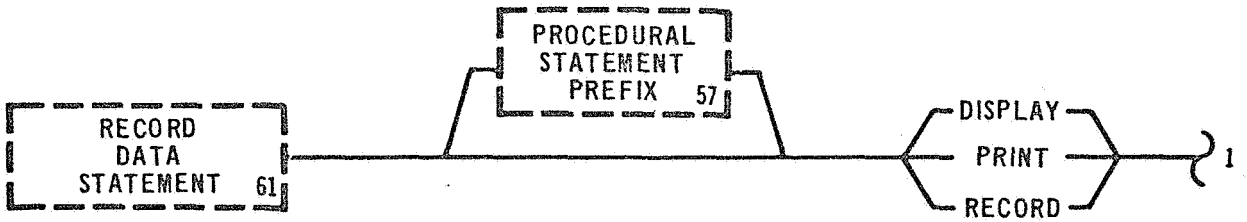
The key words "TURN ON" and "TURN OFF" are not valid for mechanical type Function Designators.

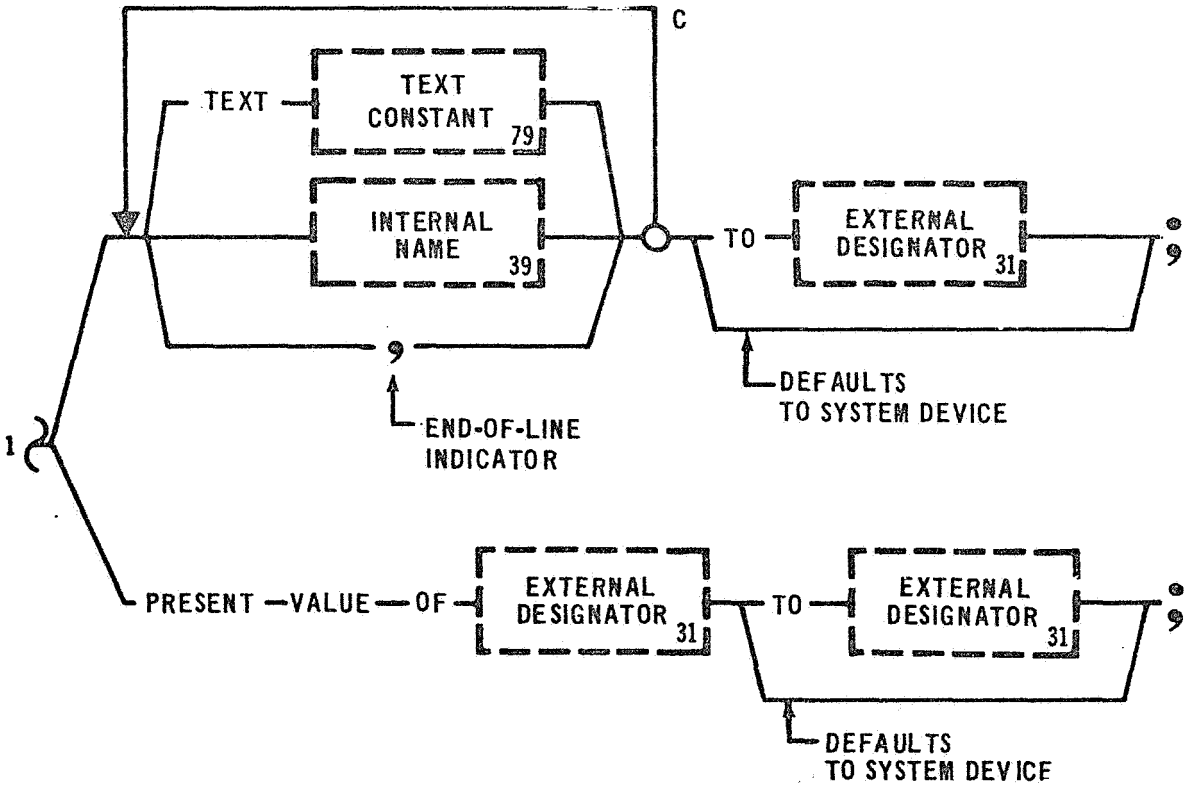
Examples: S 104 AFTER <CLOCK> IS -1 HRS, OPEN <HELIUM SUPPLY> ;

STEP 5 TURN ON (THRUST OK IND) FUNCTIONS;

61
REV 0

## RECORD DATA STATEMENT



3.1.2.4 Record Data Statement

Examples: DISPLAY TEXT (ALL SYSTEMS READY FOR POWER TRANSFER) TO

<CRT 9> ;

RECORD (INTERNAL TIME) TO < MAG 2-5 > ;

The RECORD DATA STATEMENT provides for the output of information to selected peripheral equipment. This includes formating and displaying of a message containing quantities whose values are dependent on run time conditions. Information may be recorded on a CRT display, magnetic tape, a printer, or any other recording device. The Language Processor will verify that the Function Designators referenced in the RECORD DATA STATEMENT are compatible with the key words Print and Display. When RECORD is used the output device is determined by the Function Designator. For example, the use of the key word, Print, with the Function Designator, < LINE PRINTER >, is considered a legal combination(s). The legal combination(s) is system dependent and therefore cannot be given at this time.

If several messages are specified within this statement, as provided for by the feedback loop, all messages will be recorded on all referenced Function Designators. When this statement is used to record the present value of one device to another, the processor will verify the devices are capable of inputing data to the computer and then it will generate an equivalent "READ" from the first Function Designator followed

by a "RECORD" to the corresponding Function Designator.

The comma indicates an end of line and allows for formatting of output data. The comma will not be recorded, but the data following the comma will be started on the next line.

Examples:
```
RECORD TEXT (POWER TRANSFER TEST COMPLETE),
       TEXT (INTERNAL MEASUREMENTS WERE RECORDED AS),
       TEXT (VOLTAGE=)  (INTERNAL VOLTS),
       TEXT (AMPS=)  (AMP) TO <LINE PRINTER> ,
       <LOG TAPE> ,<CRT 2> ;
RECORD PRESENT VALUE OF <CDC> TO <CRT 1> ,
       <LINE PRINTER> ;
```

### 3.1.3 Response Statements

The Response Statements are those statements that acquire data from the system under test. The test point used for the acquisition of data must be specified as a sensor type Function Designator. A Function Designator used in the REQUEST KEYBOARD STATEMENT may be either a sensor or system type.

The following statements make up the Response subgroup of the Test Action Statements:

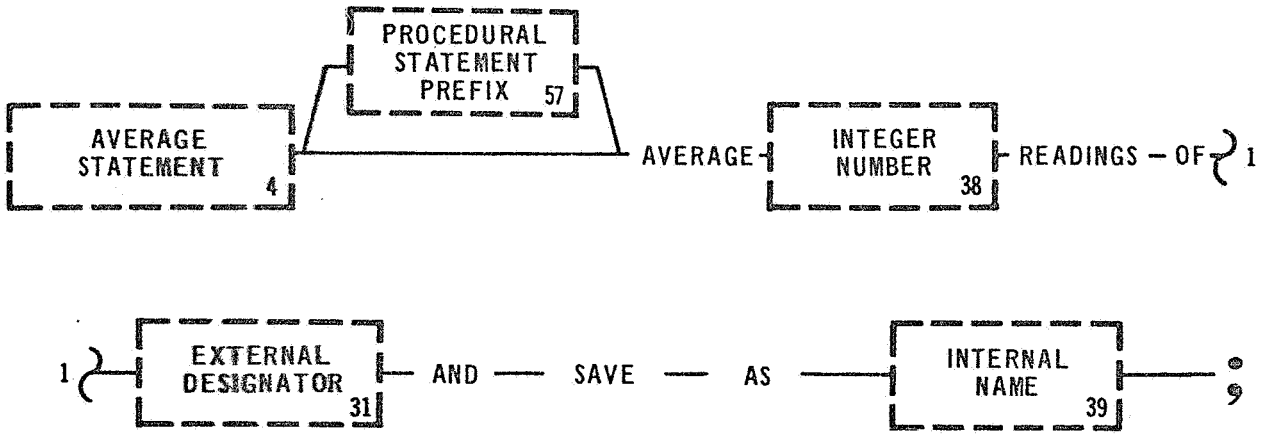
AVERAGE STATEMENT

READ STATEMENT

REQUEST KEYBOARD STATEMENT

4
REV 0

## AVERAGE STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

3.1.3.1 Average Statement

Examples: AVERAGE 10 READINGS OF < IU COOLANT TEMPERATURE > AND SAVE AS (COOLANT TEMP);

AVERAGE 5 READINGS OF < E4 HELIUM TANK P > AND SAVE AS (HE TANK PRESSURE);

The AVERAGE STATEMENT provides a convenient means of averaging a measurement or a group of measurements acquired from a system under test. The result of this statement is defined as a simple mathematical average, the sum of the measurements of a test point divided by the number of times the test point is read. The result of this statement is stored as an Internal Name. There is no provision to specify a time delay between readings. The time delta between readings is a function of the test system. If a time delay is required between measurements, the READ STATEMENT and LET EQUAL STATEMENT used in connection with the DELAY STATEMENT can be used to achieve these results. The AVERAGE STATEMENT should be used only on a measurement that is relatively stable.

If a table of Function Designators is to be read and averaged, the active Function Designator of each row is acquired the number of times indicated and averaged. Readings of Function Designators, listed in a table, are done sequentially. It is not necessary to save the results in the same table that the Function Designators are in. The average

value of each active Function Designator is stored in the identified column, in a row corresponding to its row number, if a row is not specified. The Language Processor will verify that the data characteristics of the Function Designator is compatible with the data characteristics of the Internal Name. The data characteristics of an Internal Name are described by a Declaration Statement, while the data characteristic of a Function Designator is defined in the Data Bank. An unsigned Integer is required to indicate the number of readings to be averaged.

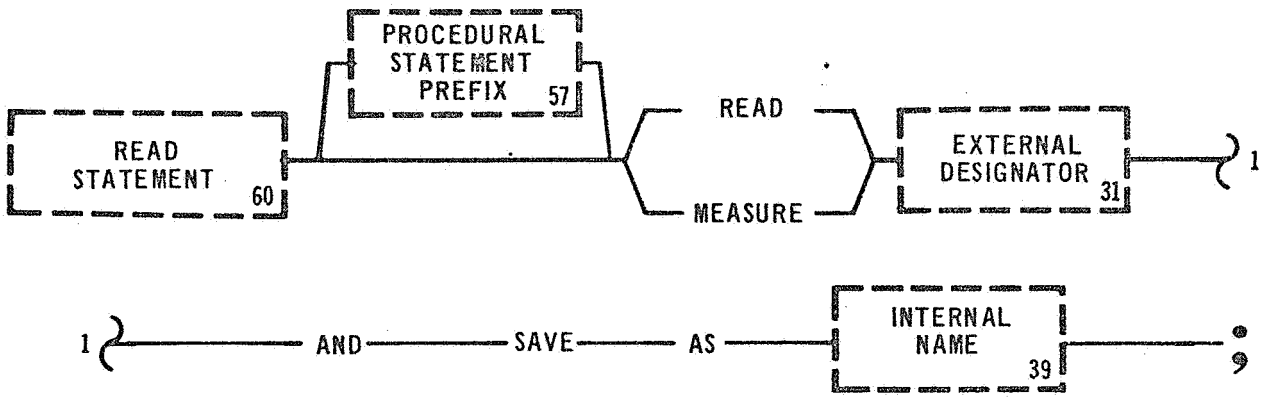
Examples: AVERAGE 10 READINGS OF (AVERAGE TABLE) AND SAVE AS COLUMN 3;

AVERAGE 20 READINGS (ANALOG TABLE) AND SAVE AS (AVERAGE TABLE) COLUMN (RESULTS);

Caution: The capability of this statement should be recognized as simple and restrictive with uncertain time deltas between readings.

60
REV 0

## READ STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

3.1.3.2 Read Statement

Examples: READ < PC STAGE INLET PRESSURE > AND SAVE AS (INLET PRESSURE);

MEASURE < IU COOLANT TEMPERATURE > AND SAVE AS (COOLANT TEMP);

The READ STATEMENT acquires data from the system under test and stores that data as internal program data. Mathematical calculations can then be performed on the internal data if desired. This data may also be used for recording purposes.

The Function Designator must be described in an active Data Bank at the time the READ STATEMENT is compiled by the Language Processor. The data characteristic of the Function Designator, as described in the Data Bank, must be compatible with the data characteristics of the Internal Name as described by a Declaration Statement.

If one Function Designator is read into a table in which only a column is specified, the data read will be stored in all the data locations in the column.

If several Function Designators, as described in a table, are read, the number of data locations must be equal to the number of Function Designators.

During the execution of a READ STATEMENT involving tables the first Function Designator listed within the table will be read. This data will be stored in the first data location in the receiving table/list. The second Function Designator will be read into the second data location in the specified column/list. This procedure continues until all Function Designators, which are active, have been read.

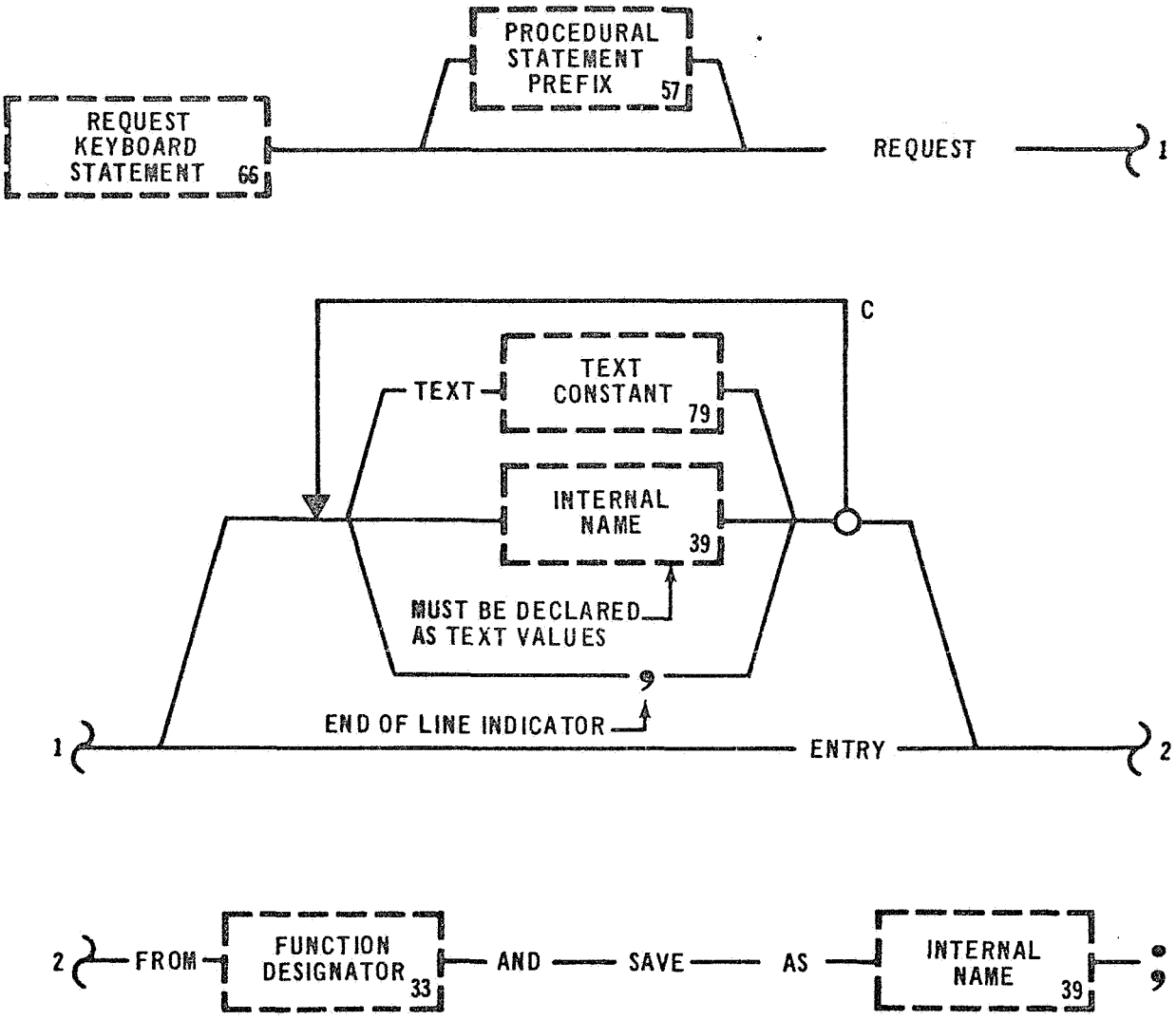
Caution: In reading data from a system under test into a table in which a column and a row are specified, the data read will be stored in the one data location. This data location will contain the data from the last activated Function Designator, listed in the table, after the execution of such a READ STATEMENT.

Examples: READ (TABLE A) FUNCTIONS AND SAVE AS (CURRENT VALUE);

MEASURE < IU COOLANT TEMPERATURE > AND SAVE AS (PRESENT STORAGE TABLE) <IU COOLANT TEMPERATURE> (ANALOG COLUMN);

66 REV 0

## REQUEST KEYBOARD STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

3.1.3.3 Request Keyboard Statement

Examples: REQUEST TEXT (DEGREES OF PITCH) FROM < CRT 7 > AND SAVE AS (DEGREES PITCH);

REQUEST ENTRY FROM < CUIP > AND SAVE AS (TM CAL MODE);

The REQUEST KEYBOARD STATEMENT allows the test procedure to request data from a test operator, and save the input for later use. One option allows a Text Constant to be displayed or printed to the test operator. A message may be inserted directly in the statement or it may be referenced by an Internal Name. Following the display of the message to the test operator, the test program will wait until the operator enters an input message. The input message will then be stored as an Internal Name, which is identified in the statement. The REQUEST ENTRY option does not allow for a message to be displayed and is generally not used in connection with a display system. This option may be used to support a teletype, test panel, etc. To accept inputs from a test panel, the statement would indicate by a light that the operator is to take some action. In this case the operator may be requested to depress a labeled light or position a switch on the test panel. This type of input will also be saved as an Internal Name.

The data characteristics of the Internal Name are defined by a Declaration Statement. In this statement the data characteristics cannot be

verified by the Language Processor for compatibility. The system will attempt to validate input data as to its characteristics.

Examples: REQUEST ENTRY FROM <TEST PANEL 5> AND SAVE AS (PANEL 5 INPUTS) COLUMN 3;

REQUEST TEXT (CAL DATA FOR ENGINE NO. 2 START TANK) AND SAVE AS (CAL TABLE) ROW 1 COLUMN 3;

## 3.2 INTERNAL SEQUENCE CONTROL

The Internal Sequence Control Statements are those statements that control the order in which statements are executed. The group of statements that make up the Internal Sequence Control Statements are:

DELAY STATEMENT

GO TO STATEMENT

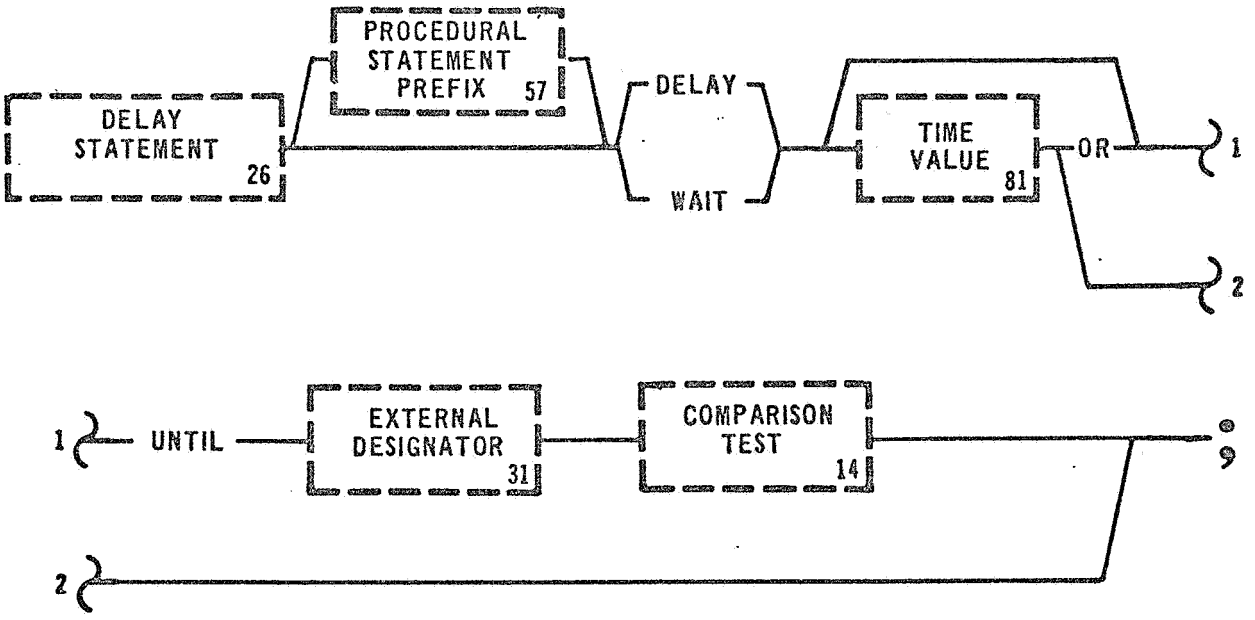
STOP STATEMENT

TERMINATE STATEMENT

REPEAT STATEMENT

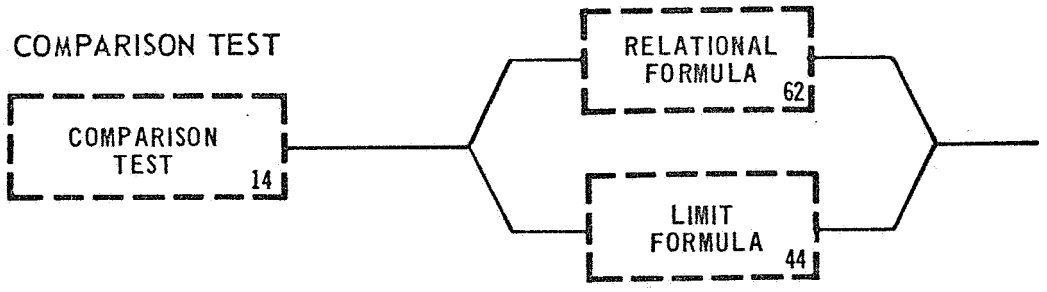
26 REV 1

## DELAY STATEMENT




14 REV 0

## COMPARISON TEST



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.2.1 Delay Statement

Examples: DELAY 5 SECS;

WAIT UNTIL < SIVB 3200 PSIA SUP VENT > IS OPEN;

The DELAY STATEMENT allows a delay in the execution of the next sequential statement. During the time the specified delay is in effect, no statements in the component, in which the DELAY STATEMENT appears, will be executed. Execution always continues with the next sequential statement.

Both conditional and unconditional delays are allowed. In the conditional delay option, the Language Processor will verify that the specified Function Designators are compatible with the type of comparison testing required. It will then generate an equivalent "READ" from the specified Function Designators. During execution the data acquired from the "READ" is then compared to the conditions specified in the Comparison Test. If the conditions are not met, the "READ" is performed again. The new data is then compared. This process continues until the conditions are met or until the Time Value has expired, provided that a Time Value was specified.

The unconditional delay option will cause a delay in the execution of a component, at least, until a specified time has expired.

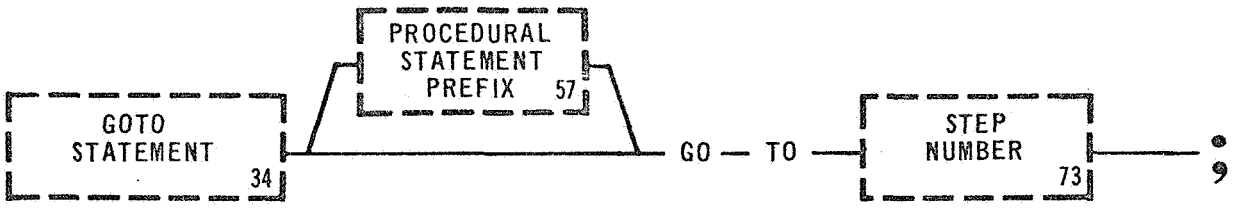
Examples: WAIT 10 SECS OR UNTIL <GN 750 PSIA BLEED VLV> IS
CLOSED;

DELAY 5 MIN OR UNTIL <CLOCK> IS EQUAL TO -3 HRS 30
MINS;

LL

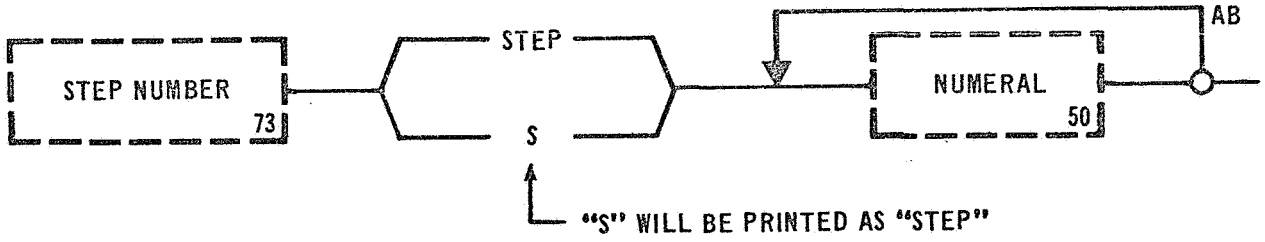
34 REV 0

## GOTO STATEMENT




73 REV 1

## STEP NUMBER



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.2.2 Go To Statement

Examples: GOTO S20;

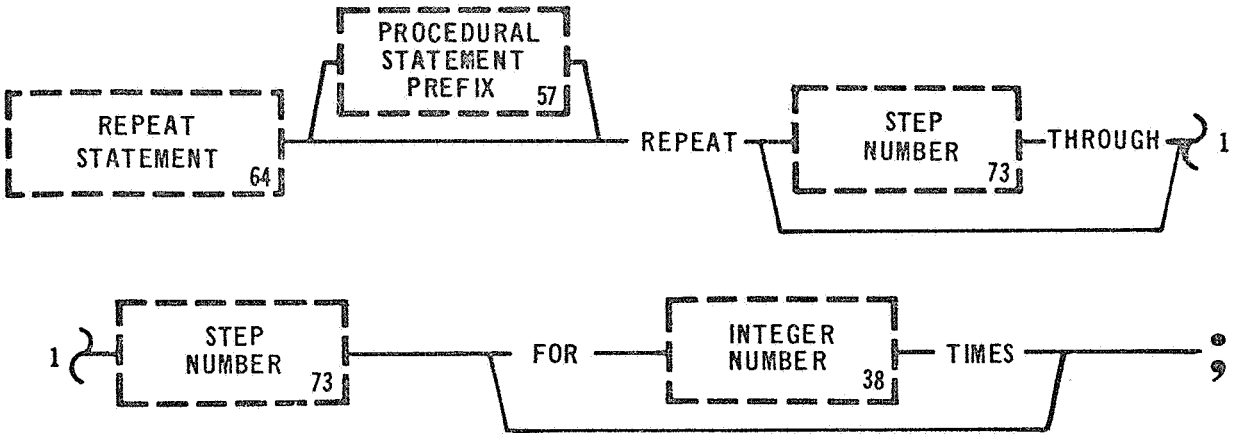
GO TO STEP 30;

The GO TO STATEMENT provides unconditional branching within a test program. After executing a GO TO STATEMENT, control is passed to the statement that is referenced by the Statement Number, following the verb. A GO TO STATEMENT is generally used at the end of a sequence of statements that represent a logical blocking of a test function, and acts to direct the computer to some other portion of the test program. Statement Numbers specified by a GO TO STATEMENT must be defined within the boundaries of a GOAL component; i.e., "local" to the component. A GO TO STATEMENT cannot be used to transfer control to a statement that is outside the boundaries of the component, as in a subroutine.

The letter "S" may be used as an abbreviation for the word STEP and it will be printed as STEP by the Language Processor.

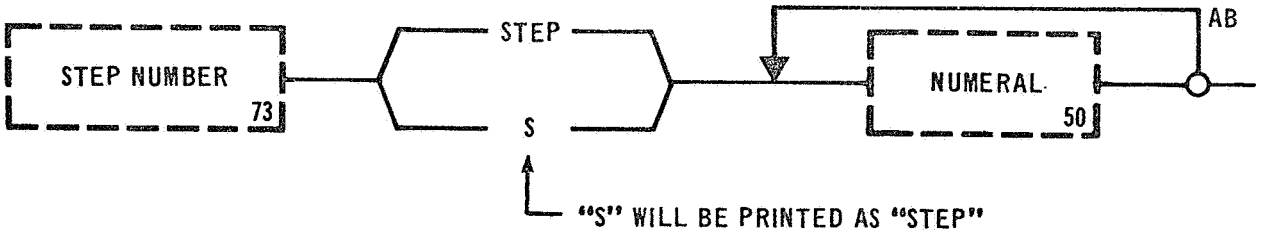
64 REV 0

## REPEAT STATEMENT




73 REV 1

## STEP NUMBER



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.2.3 Repeat Statement

Examples: REPEAT STEP 30;

REPEAT STEP 5 THROUGH STEP 7;

The REPEAT STATEMENT allows repetition of a single statement or a group of statements. Any single Procedural Statement with a Statement Number may be repeated. The statement may be repeated as many times as necessary as indicated by an unsigned Integer number. Any combination of Procedural Statements in a sequence may be repeated. The only requirement is that the first and last statements of the sequence must have Statement Numbers. The first Statement Number and the second Statement Number in the REPEAT STATEMENT diagram must reference the first and last statements of the repeat loop respectively. The first and last statements form the boundaries of a repeat loop. Statements within the repeat loop will be executed in the normal fashion. Branching may be used inside a repeat loop. If branching is used to transfer control outside the repeat loop, the REPEAT STATEMENT will lose control at this point and the effect of the repeat will be cancelled. The next time the last statement in the repeat loop is executed, the original REPEAT STATEMENT will have no effect. A REPEAT STATEMENT may be embedded in a repeat loop to form a nested repeat loop. The sequence of statements to be repeated can either precede or follow the REPEAT STATEMENT. A REPEAT STATEMENT cannot appear

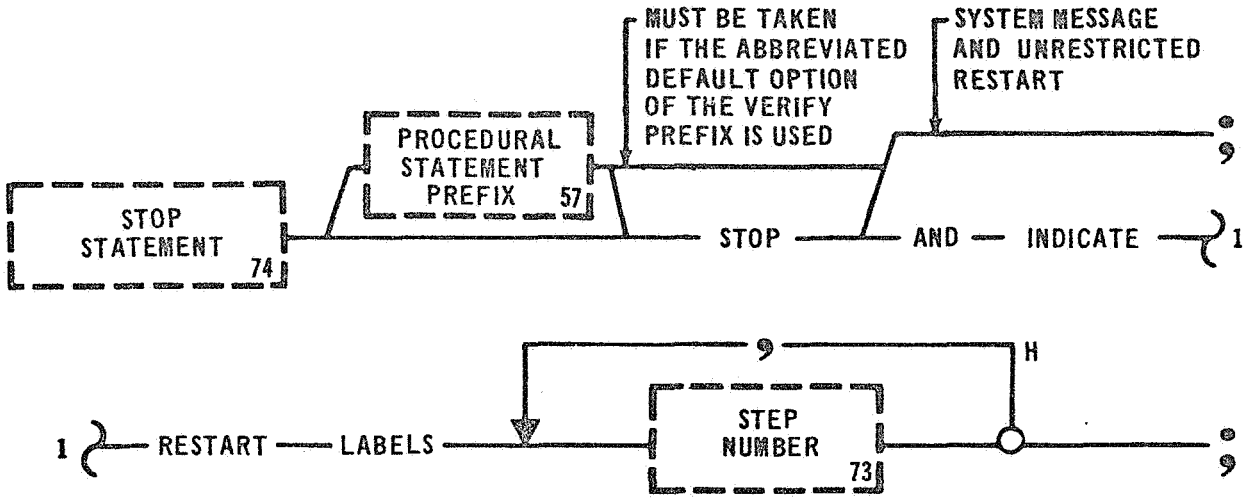
within the bounds of the repeat grouping it initiates. Also, a REPEAT STATEMENT cannot reference itself.

Examples: REPEAT STEP 1000 THROUGH STEP 1100 FOR 20 TIMES;

VERIFY < MAIN POWER > IS ON ELSE REPEAT STEP 540 THROUGH STEP 550;

74 REV 1

## STOP STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.2.4 Stop Statement

Examples: STOP;

STOP AND INDICATE RESTART LABELS S100, S200;

The STOP STATEMENT provides manual intervention capability during execution of a test program. The STOP STATEMENT has two primary options; the restricted and the unrestricted restart points of a test program. The restricted mode allows the test program to be restarted only at preselected points. The preselected points are determined by the test engineer at coding time. During the execution of this statement the program will stop execution after it has displayed a tutorial message to the test operator indicating the restricted mode and the restart options that are available. The test operator then has the option of selecting one restart point from the indicated restart points.

The unrestricted mode allows the test program to be restarted at any Procedural Statement within the test procedure. With this option the test writer does not control the restart point at coding time. Extreme caution should be used when incorporating this option in a test program. Arbitrary branching within a test program could have a serious effect on the system under test or the test program.

During execution of this option an appropriate tutorial message will be displayed to the test operator. The test program will stop execution,

except for concurrent type processing until the console operator takes the appropriate action.

The STOP STATEMENT also allows the use of an abbreviated form when combined with the Verify option of the Verify Prefix.

VERIFY < MAIN POWER > IS ON ELSE RECORD EXCEPTION AND STOP;

This statement invokes the default option for system messages and system devices. The abbreviated form of the above example is VERIFY < MAIN POWER > IS ON;

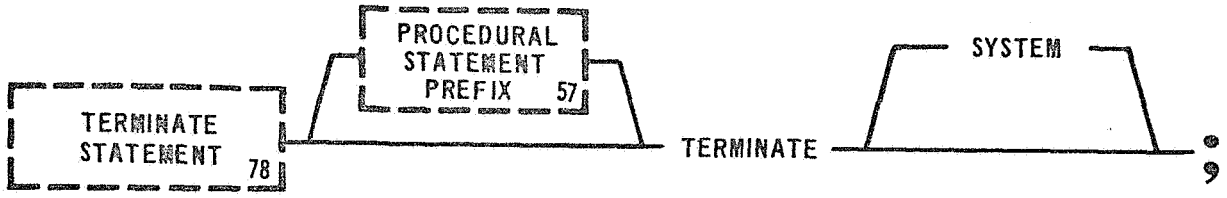
Before using this abbreviated version the user must be thoroughly familiar with the default condition and understand this is an unrestricted STOP STATEMENT.

Examples: STOP AND INDICATE RESTART LABELS STEP 10, S20, STATEMENT 300;

VERIFY < POWER SELECTOR > IS ON ELSE DISPLAY EXCEPTION TO < CRT 2 > AND STOP AND INDICATE RESTART LABELS S5, S10;

78
REV 0

## TERMINATE STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.2.5 Terminate Statement

Examples: TERMINATE;

TERMINATE SYSTEM;

The TERMINATE STATEMENT causes cessation of execution of a program or subroutine. There are two options: the Terminate and the Terminate System. The Terminate option is discussed first. If a subroutine, containing a Terminate option, is called into execution by a PERFORM SUBROUTINE STATEMENT, when the Terminate option is executed it will cause control to be returned to the calling program. Control will be returned to the next sequential statement after the PERFORM SUBROUTINE STATEMENT. If a subroutine is called into execution by an Interrupt as defined by the WHEN INTERRUPT STATEMENT, the Terminate option will cause control to be passed back to the calling program as directed by the WHEN INTERRUPT STATEMENT.

A Terminate option within a program will, when executed, stop the execution of the test program. If a program is called into execution by a PERFORM PROGRAM STATEMENT, the Terminate option will stop execution of the program and control will be returned to the program which called it. Control is always returned to next sequential statement after the PERFORM PROGRAM STATEMENT.

In a concurrent program, the Terminate option will stop execution of the program. Control is not passed back to the calling program but is passed to the system executive. In a cyclic execution mode of a concurrent program the Terminate option will stop the execution of the concurrent program. The concurrent program will be restarted at the beginning of the next cycle. The calling program is executed independent of the concurrent program and is not affected by the Terminate option contained in the concurrent program.

The Terminate System option provides for a complete GOAL application program system shutdown. If a subroutine is called into execution by a PERFORM SUBROUTINE STATEMENT or an interrupt, a TERMINATE SYSTEM STATEMENT will cause the execution of subroutine to terminate. The calling program will also be terminated. If a TERMINATE SYSTEM STATEMENT is executed in a test program, called into execution by a PERFORM PROGRAM STATEMENT, program execution will be stopped. Execution of the calling program will also be stopped.

In a concurrent program the Terminate System option will stop execution of it. In a cyclic concurrent program, the Terminate System option will, when executed, stop the execution of the concurrent program. The concurrent program will not be restarted at the beginning of the next cycle.

## 3.3 ARITHMETIC/LOGICAL OPERATIONS

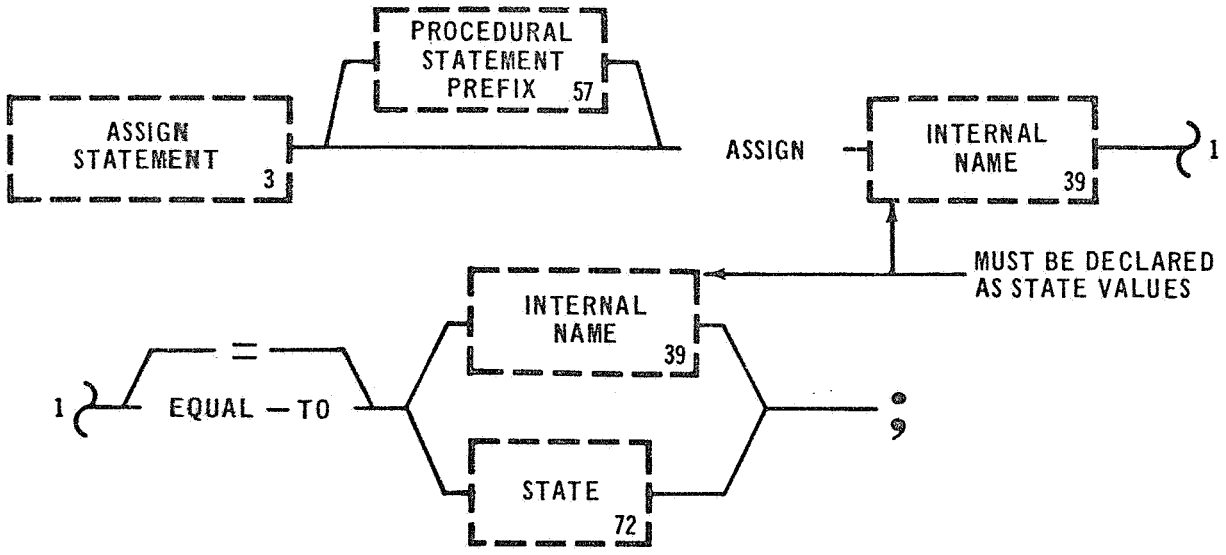
The Arithmetic and Logical Statements are those statements that provide the mathematical computation, and logical operations allowed in the language. This group consists of the following statements:

ASSIGN STATEMENT

LET EQUAL STATEMENT

3
REV 0

## ASSIGN STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.3.1 Assign Statement

Examples: ASSIGN (FLAG B) = ON;

ASSIGN (STAGE POWER) EQUAL TO (FLAG B);

The ASSIGN STATEMENT assigns a state to an Internal Name. This is available for setting "flags" which may be used for internal sequence control within a test program. It may also be used to change any State declaration for use in other GOAL statements. The ASSIGN STATEMENT has two options: Internal Name and State. During the execution of the State option the State specified on the right will replace the condition of the Internal Name of the left. If the Internal Name option is used instead of the State option, the state of the Internal Name on the right side will replace the condition of the Internal Name on the left side.

The Language Processor will ensure that any Internal Name used by an ASSIGN STATEMENT has been declared as a State value.

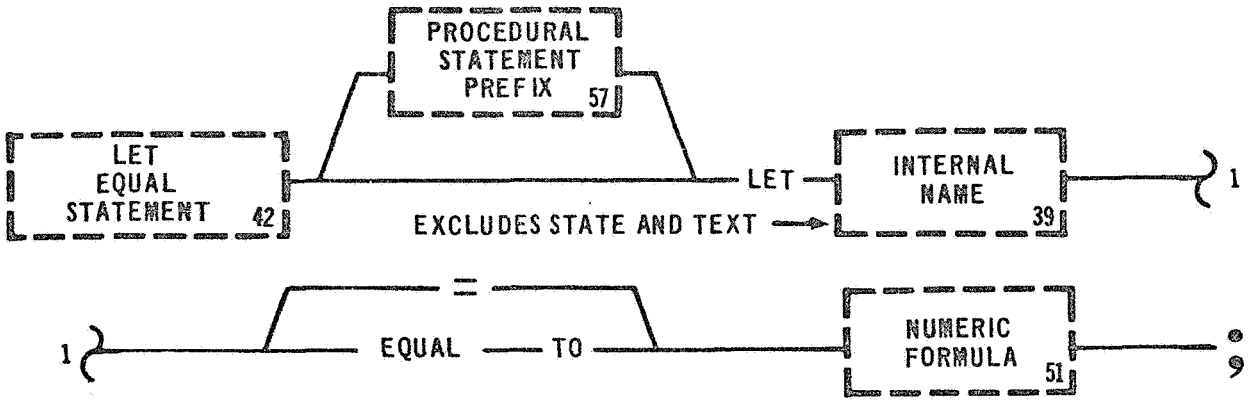
Examples: VERIFY <CLOCK> IS GREATER THAN -3 MINS 19 SEC THEN

ASSIGN (INTERNAL SEQ) EQUAL TO ON;

STEP 20 IF (A) = 0, ASSIGN (FLAG A) = OFF;

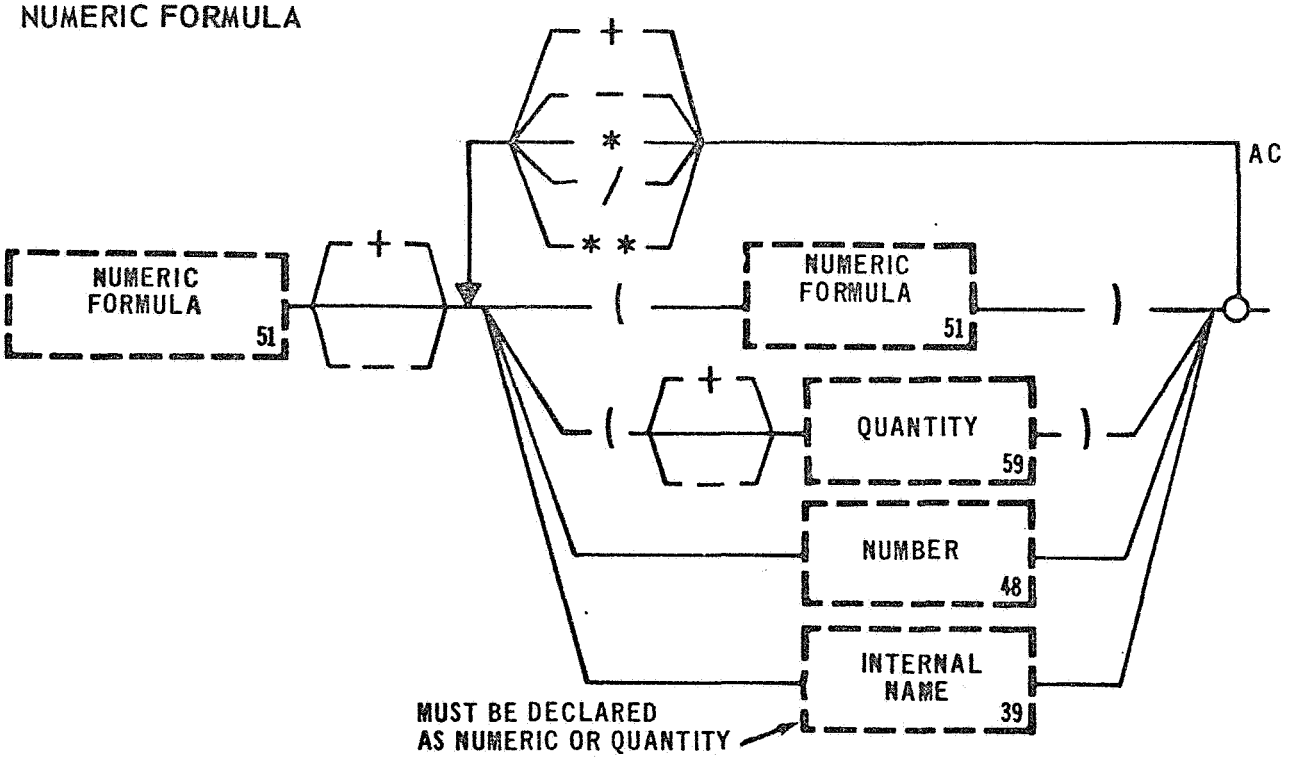
42 REV 0

## LET EQUAL STATEMENT




51 REV 0

## NUMERIC FORMULA



### 3.3.2 Let Equal Statement

Examples: LET (A) = (A) + 1;

LET (B) = (A);

The LET EQUAL STATEMENT provides for the assignment of numerical values to Internal Names through the use of the Numeric Formula. This statement closely resembles conventional arithmetic formulas and is used to perform necessary mathematical calculations. It should be noted that all Internal Names or variables are delimited by parentheses. The numeric calcualtion to be performed by a test program is defined by the Numeric Formula. In the LET EQUAL STATEMENT the equal sign or the words "EQUAL TO" means "is replaced by" rather than "is equivalent to". During execution the results of the Numeric Formula replaces the contents of the Internal Name defined on the left side of the equation. For example: LET (A) = (A) + 1; means that the Internal Name (A) is to be replaced with the old value plus one. Quantities may be mixed as desired. The hierarchy of operations is discussed in the Numeric Formula writeup.

```
Examples: LET (SINX) = (XR) - ((XR) ** 3/6) + ((XR) ** 5/120) -
              - ((XR) ** 7/5040) ;
          LET (X) = (((((B) ** 2) - (4 * (A) * (C)))** 0.5) - (B))
              / 2 * (A);
```

## 3.4 EXECUTION CONTROL

The Execution Control Statements regulate the execution of a specified component. This group of statements is necessary to control the mode of execution of a program, subroutine, or monitor type statements. The Execution Control group consists of the following statements:

CONCURRENT STATEMENT

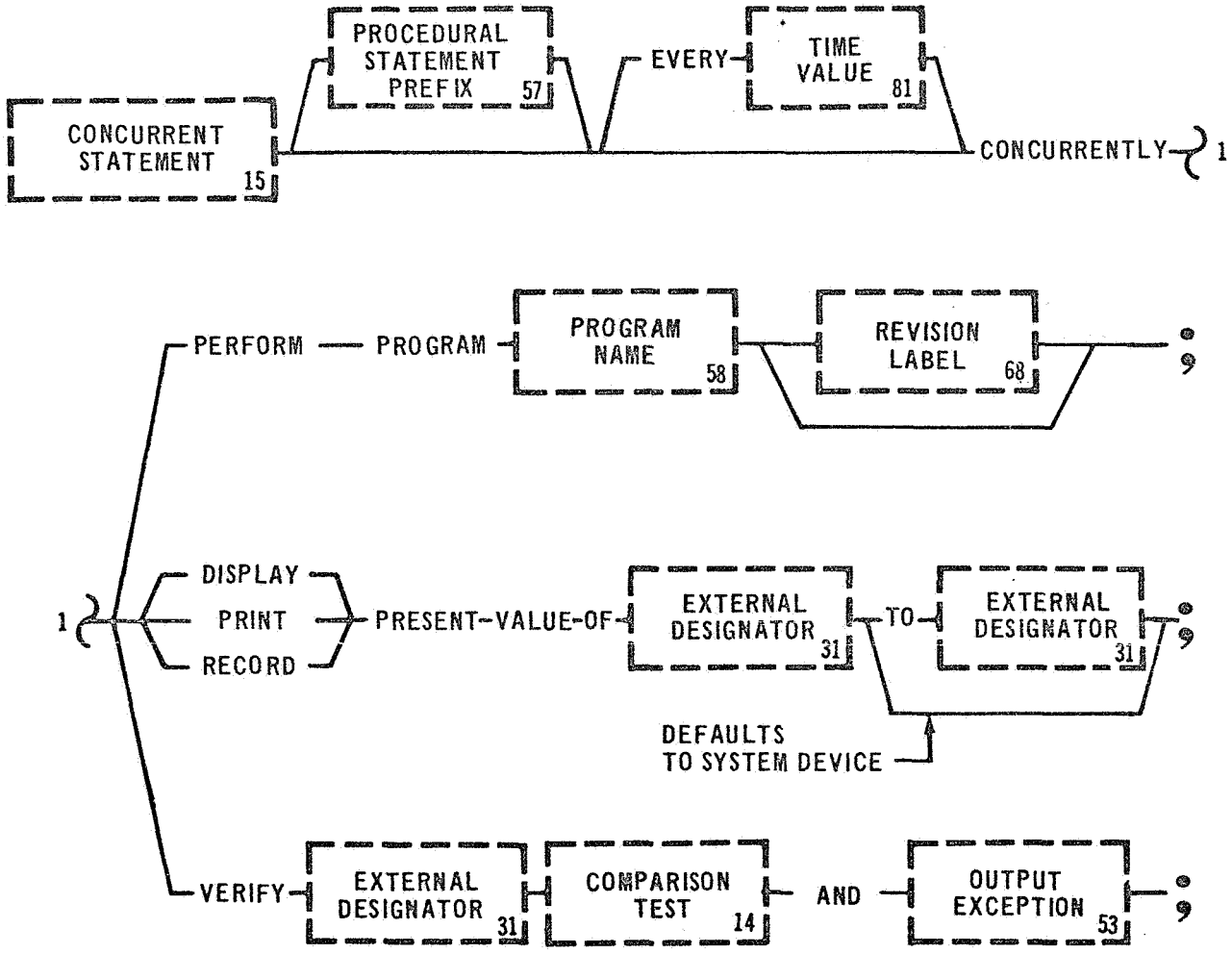
RELEASE CONCURRENT STATEMENT

PERFORM PROGRAM STATEMENT

PERFORM SUBROUTINE STATEMENT

15
REV 1

## CONCURRENT STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

3.4.1 Concurrent Statement

Examples: CONCURRENTLY PERFORM PROGRAM (BE01) REVISION 10;

CONCURRENTLY DISPLAY PRESENT VALUE OF <E1 MAIN FUEL FLOW> TO <CRT 15> ;

CONCURRENTLY VERIFY <PRESS ENG 102 GIMBAL> IS BETWEEN 1665 PSIA AND 1465 PSIA AND DISPLAY EXCEPTIONS TO <CRT 12> ;

The CONCURRENT STATEMENT allows parallel operations. This statement has three main options: the Perform Program, Record, and Verify options.

The Perform Program option allows for concurrent execution of a test program. A Time Value may be used to achieve a cyclic execution of a concurrent program. If the specified time interval expires before execution is finished, the concurrent program will continue normal execution. The concurrent program will then be restarted immediately after completion. If the execution of a concurrent program is completed before the time interval has expired, the restart of the concurrent program will be delayed until the time interval expires. The concurrent program will be executed only once, if cyclic rate is not specified.

A TERMINATE STATEMENT, within the concurrent program will stop the execution of a concurrent program. It will not halt the cyclic

execution of the concurrent statement. The cyclic execution is stopped by either a TERMINATE STATEMENT, System option in the concurrent test program, or a Release Statement in the calling test program.

The Record option provides a data monitoring capability. A Time Value may be used to specify the monitoring rate. The present value of the first Function Designator will be recorded on the second Function Designator, computer peripheral equipment. If a group of test points is monitored they will be recorded on all indicated peripheral devices. A reading from a test point may be recorded on any combination of peripheral devices.

The Verify option provides an exception monitor capability. A Time Value may be used to specify the monitor rate. The data is limit checked every cycle but no action is taken unless the data fails the Comparison Test. An out of tolerance condition can force an error message to be output to the test operator. Out of tolerance data may be recorded on any available peripheral equipment.

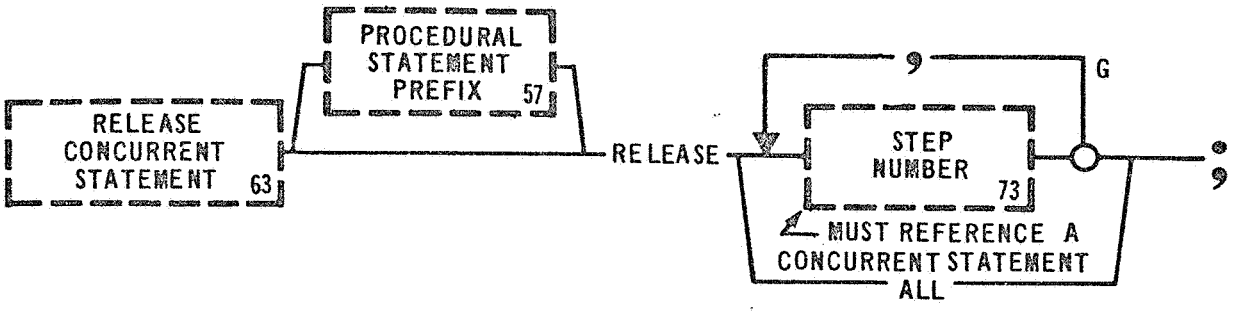
Examples: IF (TIME) IS -3 MIN 31 SEC THEN EVERY 2 SEC CONCURRENTLY
PERFORM PROGRAM (NT60) REVISION 1;
EVERY 1 SEC CONCURRENTLY RECORD PRESENT VALUE OF <TEMP.
HE INLET VALVE> ,<TEMP. GAS. HELIUM TANK> TO
<LOG TAPE 2-5> ,<CRT 12> ,<CRT 13> ;
STEP 20 EVERY 2 SEC CONCURRENTLY VERIFY <VOLT-OUTPUT
AFT BATTERY NO 1> ,<VOLT OUTPUT AFT BATTERY
NO 2> , IS BETWEEN 61.0V and 51.0V AND RECORD
EXCEPTION (BATTERY VOLTAGE IS OUT OF TOLERANCE)

Examples: (CTD)

TO <LINE PRINTER 1> ,<CRT 13> ;

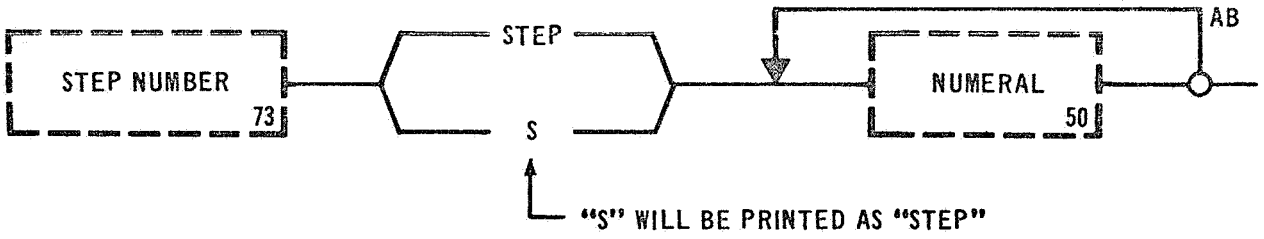
63 REV 0

## RELEASE CONCURRENT STATEMENT




73 REV 1

## STEP NUMBER



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.4.2 Release Concurrent Statement

Examples: RELEASE STEP 10;

RELEASE ALL;

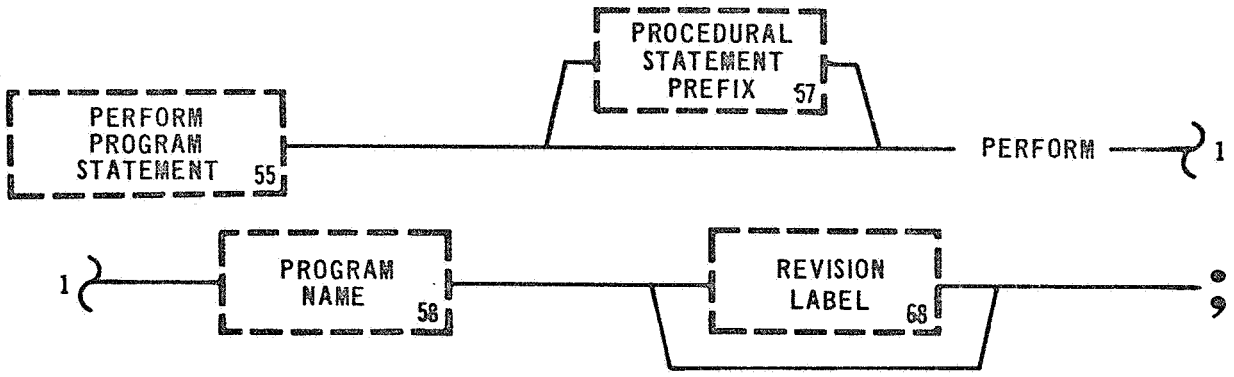
The RELEASE CONCURRENT STATEMENT releases from execution status the parallel operations initiated by a CONCURRENT STATEMENT. The RELEASE CONCURRENT STATEMENT will terminate the cyclic action of those CONCURRENT STATEMENTS which are referenced by a Statement Number. The Language Processor will verify that the Statement Number used to release a CONCURRENT STATEMENT references a CONCURRENT STATEMENT. The ALL option allows for releasing of all CONCURRENT STATEMENTS activated by the component in which the RELEASE CONCURRENT STATEMENT resides. The RELEASE CONCURRENT STATEMENT can release only the concurrent operations started within the same component.

Examples: STEP 440 RELEASE STEP 10, STEP 20, STEP 16;

VERIFY < E1 RECIRC PUMP > IS OFF THEN RELEASE ALL;

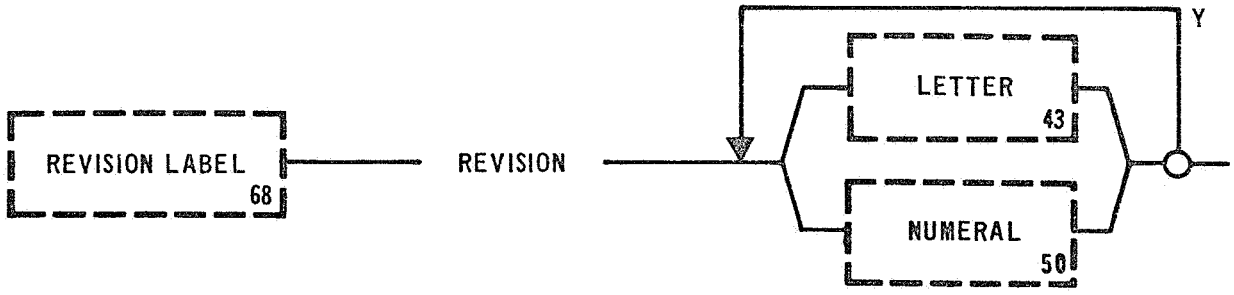
55 REV 0

## PERFORM PROGRAM STATEMENT




68 REV 0

## REVISION LABEL



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.4.3 Perform Program Statement

Examples: PERFORM PROGRAM (LVDC POWER ON);

PERFORM PROGRAM (ATTITUDE COMMAND TEST) REVISION 03;

The PERFORM PROGRAM STATEMENT initiates execution of a specified program. The execution of the calling test program stops when it starts another program by the PERFORM PROGRAM STATEMENT. When a TERMINATE STATEMENT or an END STATEMENT is encountered within the second level program, control is passed back to the calling program. Execution of the calling program is then picked up at the next statement following the PERFORM PROGRAM STATEMENT. Communication between different levels of programs can be achieved by the use of a Function Designator that is predefined as a system type.

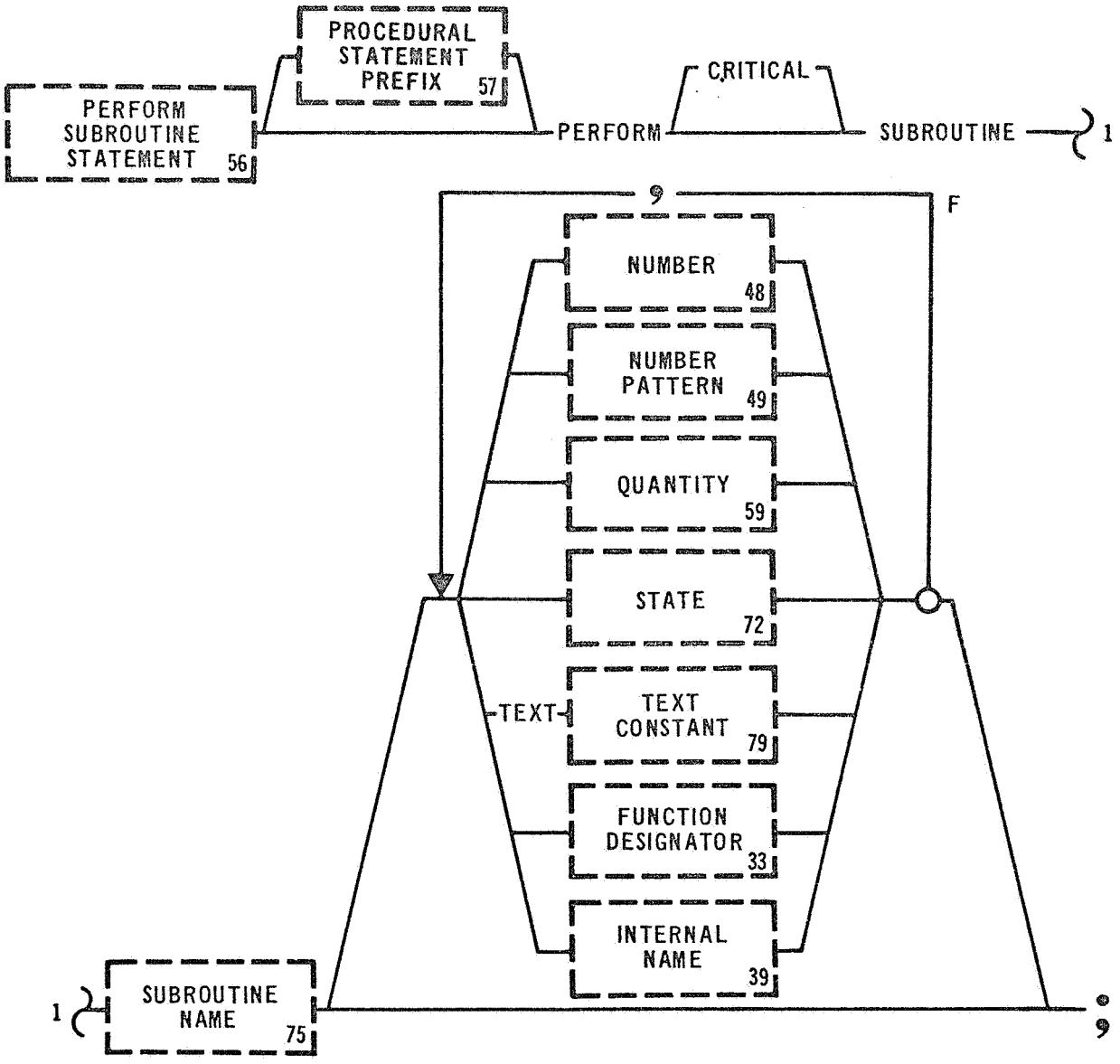
The Language Processor cannot verify the existence or the uniqueness of the Program Name referenced in the PERFORM PROGRAM STATEMENT. The executive must have access to the referenced program name at execution time. During execution the Program Name is used to identify the other program. The Revision Label may be used to select the proper configuration level of a particular program. If used by the PERFORM PROGRAM STATEMENT, the Revision Label in effect becomes part of the name the executive uses in the retrieval operation.

Examples: WHEN<CLOCK> = -10 HRS THEN PERFORM PROGRAM (VM-GSE PREPS);

STEP 252 AFTER<CLOCK> IS -3 MIN, VERIFY<TERMINAL SEQUENCE> IS ON THEN PERFORM PROGRAM (REDLINE MONITOR) REVISION 020;

56
REV 0

## PERFORM SUBROUTINE STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.4.4 Perform Subroutine Statement

Examples: PERFORM CRITICAL SUBROUTINE (CALCULATE DELAY TIME);

PERFORM SUBROUTINE (TEST VOLTAGE);

The PERFORM SUBROUTINE STATEMENT initiates execution of a previously defined subroutine. A subroutine may be performed in two modes, critical and noncritical. The term critical indicates that the actions being performed in the subroutine are time dependent and, as such, should not be interrupted. It insures a basic amount of central processor time for the execution of the subroutine. Language Level Interrupts are not allowed to break into the execution of a critical subroutine. Communication between the calling program and a subroutine may be achieved by passing substitutable data at execution time. The substitutable data defined in the PERFORM SUBROUTINE STATEMENT will, at execution time, replace the contents of the Parameters defined in the BEGIN SUBROUTINE STATEMENT. The contents of the Parameters are available as both inputs and outputs of the subroutine. The number of Parameters declared in the BEGIN SUBROUTINE STATEMENT must equal the number of substitutable data items.

All names and Statement Numbers within the subroutine are local to the subroutine. Therefore a Statement Number when used to reference a statement must reference a statement within the subroutine. All names used within a subroutine must be declared in the subroutine. Subroutine

parameter passing is discussed in Section V.

Examples: VERIFY<LVDC> POWER IS ON THEN PERFORM CRITICAL SUBROUTINE (LVDC SUMCHECK) T 7777777, T0000000;

AFTER<CLOCK> IS -22 MINS THEN IF (POWER TRANSFER FLAG) IS ON THEN PERFORM SUBROUTINE (POWER TRANSFER) 24.0V, +0.5V,-0.5V,10SEC;

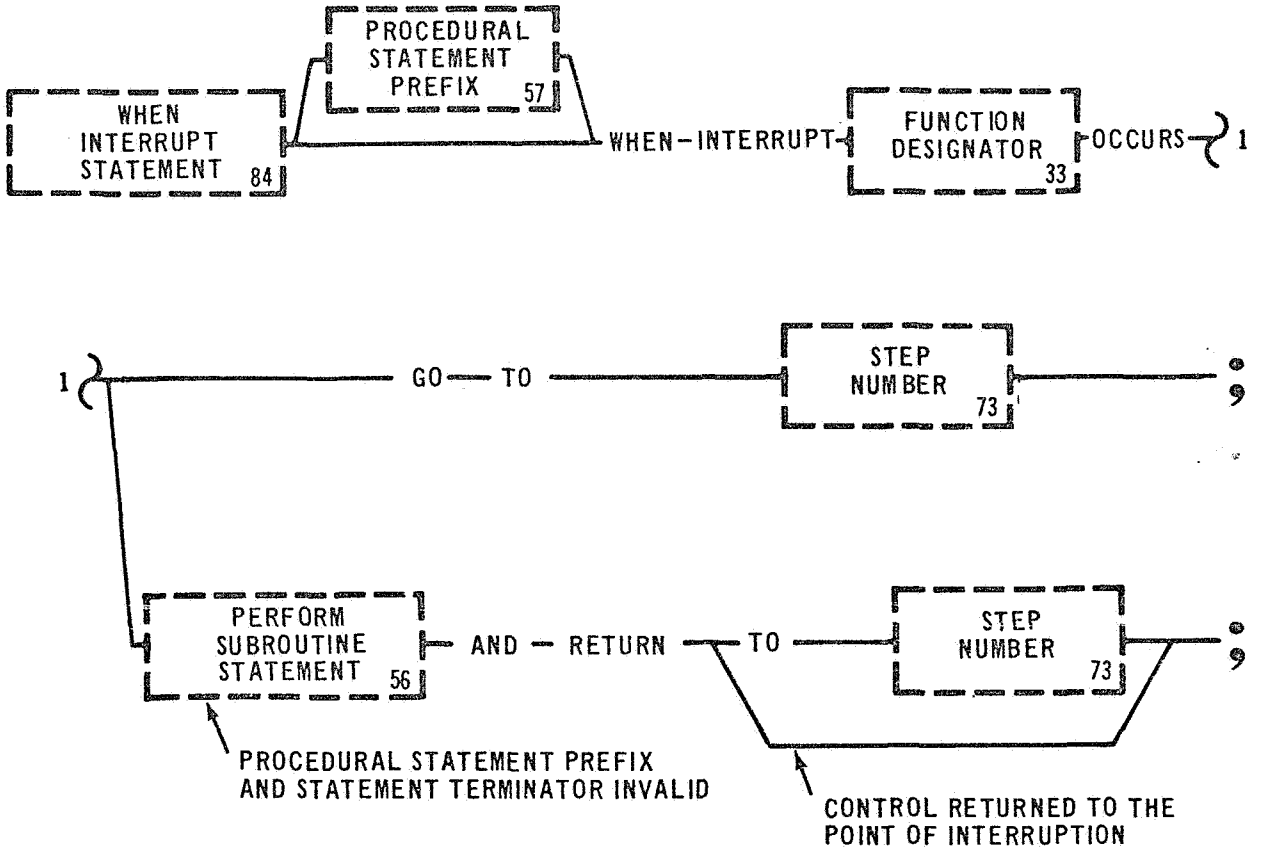
## 3.5 INTERRUPT CONTROL

The Interrupt Control Statements provide assignment of an interrupt status. Interrupts, as defined in the test program, can only be honored when they are in an active status. The following statements make up the Interrupt Control group:

WHEN INTERRUPT STATEMENT

DISABLE INTERRUPT STATEMENT

84 REV 1 WHEN INTERRUPT STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

3.5.1 When Interrupt Statement

Examples: WHEN INTERRUPT < POWER FAILURE > OCCURS GO TO STEP 9000;

WHEN INTERRUPT < MAIN POWER > OCCURS GO TO STEP 333;

The WHEN INTERRUPT STATEMENT enables a Language Level Interrupt. The Function Designator identified as an interrupt, by this statement, must also be specified in the Data Bank as an interrupt. The conditions necessary for the occurrence of the interrupt are also defined within the Data Bank. The WHEN INTERRUPT STATEMENT has two options: the Perform Subroutine and the Go To. During the execution of this statement the specified interrupt is activated, but the referenced subroutine or Go To option is not executed, at this point. After execution of this statement, the remaining Procedural Statements are executed in their normal sequence.

The Go To option is discussed first. If the interrupt conditions are met, the normal execution of the test program is stopped. Control is then passed to the statement identified in the Go To option of the WHEN INTERRUPT STATEMENT. The test program continues normal execution from that point. If the Perform Subroutine option was selected and the interrupt conditions are met, the normal execution of the test program is stopped. Control is passed to the indicated subroutine. If a TERMINATE STATEMENT is then encountered, control will be returned to the test program as directed by the WHEN INTERRUPT STATEMENT. If

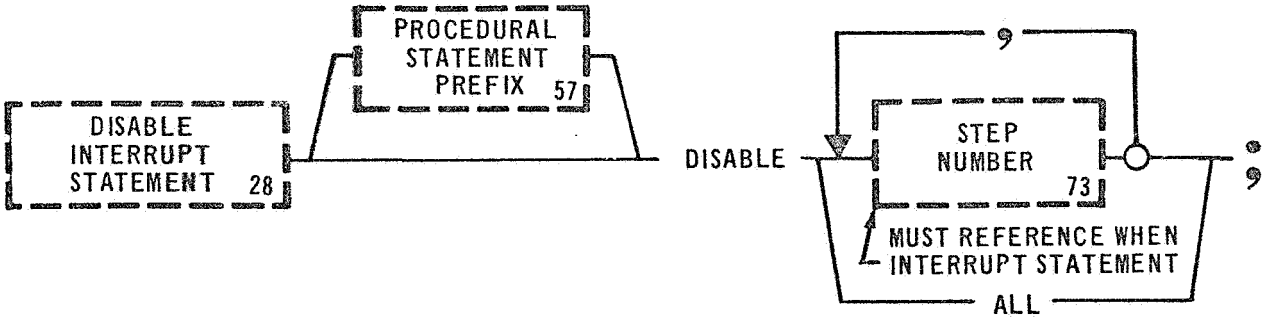
the statement does not specify a Statement Number it will be returned to the calling program at the point it left to process the interrupt.

If a TERMINATE STATEMENT is not encountered, execution of the subroutine control will be returned as indicated by the WHEN INTERRUPT STATEMENT.

Examples: IF (SUBROUTINE A FLAG) IS ON THEN WHEN INTERRUPT< HY DR
PUMP PRESS > OCCURS PERFORM SUBROUTINE (PUMP CHECKS)
20 PSIA, ON, OFF AND RETURN;
WHEN INTERRUPT< CLOCK T-22 MINS > OCCURS PERFORM SUBROUTINE
(START TANK CHILLDOWN) AND RETURN TO STEP 9999;

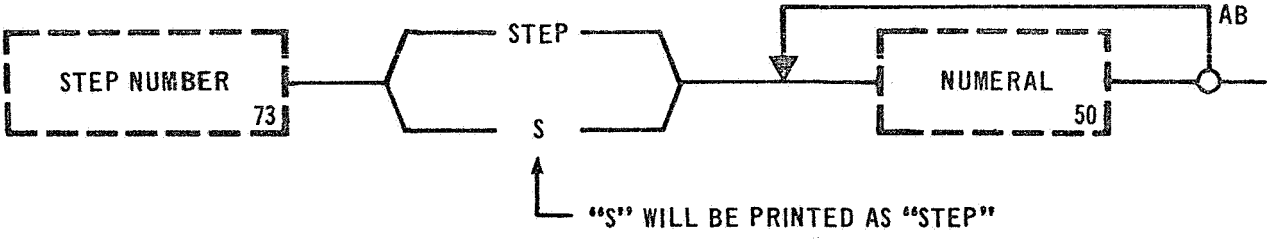
28
REV 0

## DISABLE INTERRUPT STATEMENT




73
REV 1

## STEP NUMBER



### 3.5.2 Disable Interrupt Statement

Examples: DISABLE STEP 20;

DISABLE ALL;

The DISABLE INTERRUPT STATEMENT disables the interrupt enabled by the referenced WHEN INTERRUPT STATEMENT. Thus, this statement is the complement of the WHEN INTERRUPT STATEMENT. The Language Processor will verify that the Statement Number references a WHEN INTERRUPT STATEMENT. The execution of a DISABLE INTERRUPT STATEMENT will result in a no operation if the referenced WHEN INTERRUPT STATEMENT has not been previously executed. A no operation will not stop the execution of the test program; it will continue in its normal sequence. The ALL option will disable all active WHEN INTERRUPT STATEMENTS used by the component in which the DISABLE INTERRUPT STATEMENT is found.

All interrupts are disabled at the start of a program and remain disabled until activated by the WHEN INTERRUPT STATEMENT. The active or disable status is valid only for the component in which it is defined. The active or disable status, of a test program that calls into execution a second program, is not valid for the second program. Interrupts activated in the second program are active only for that program. When the second program passes control back to the calling program, the original status of the interrupts are reinstated. Interrupts enabled by a program are suspended during execution of a subroutine. Language

Level Interrupts are discussed in Section V.

Examples: DISABLE STEP 333;

VERIFY<MAIN POWER> IS OFF THEN DISABLE STEP 55, STEP 70;

## 3.6 TABLE CONTROL

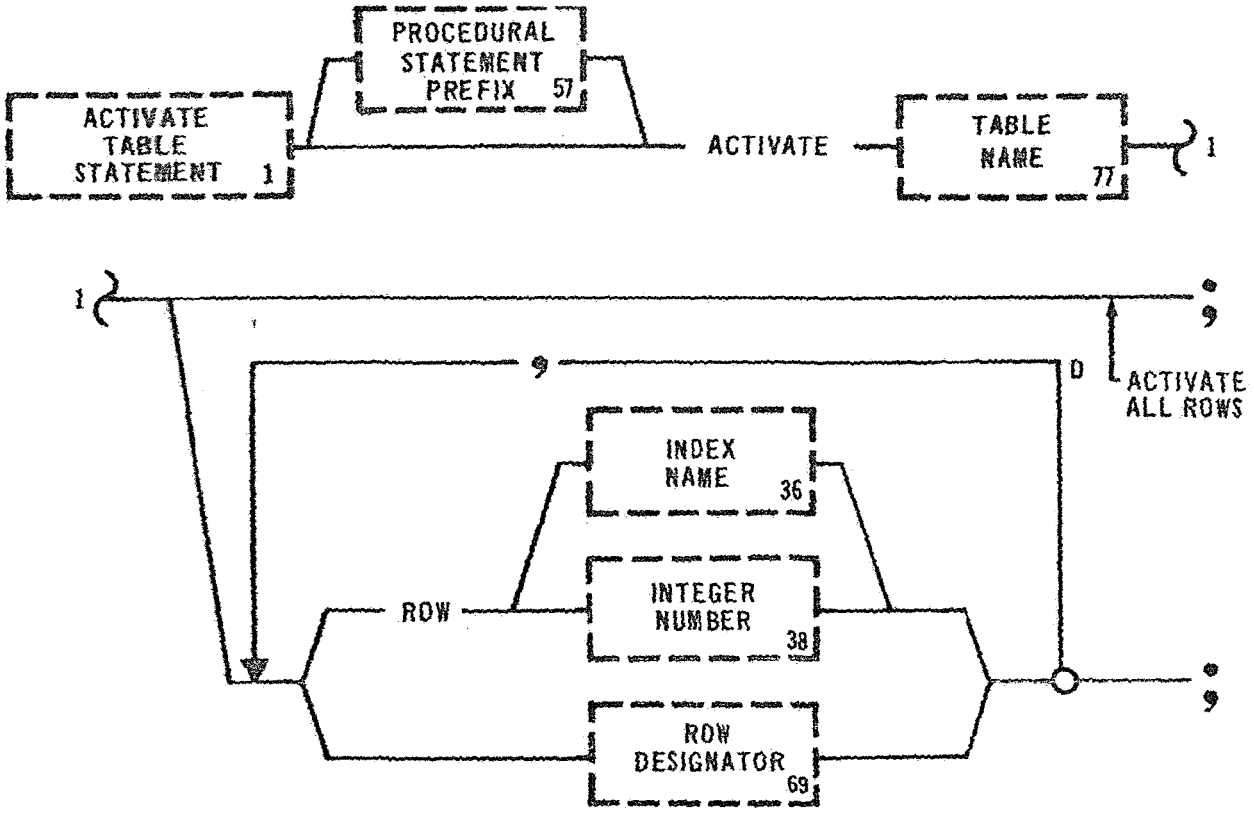
The Table Control Statements assign an active or inactive status to individual rows of a specified table. The following statements make up the Table Control Statements:

ACTIVATE TABLE STATEMENT

INHIBIT TABLE STATEMENT

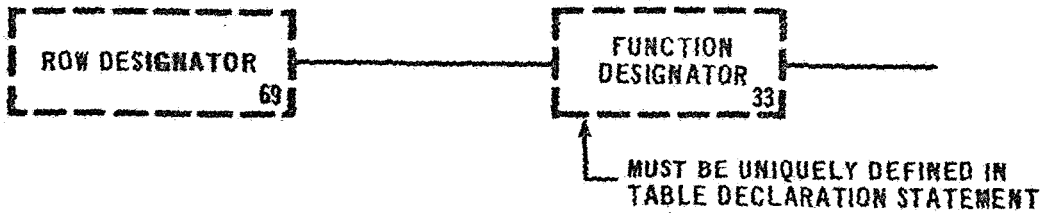
1 REV 0

## ACTIVATE TABLE STATEMENT




69 REV 0

## ROW DESIGNATOR



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.6.1 Activate Table Statement

Examples: ACTIVATE (DISCRETE TABLE);

ACTIVATE (TABLE A) ROW 1, ROW 2, ROW 3;

The ACTIVATE TABLE STATEMENT assigns an active status to a table or individual rows of a table. The ACTIVATE TABLE STATEMENT deals only with the Function Designators and not the data positions. System under test access via a table can only be performed on active Function Designators. All rows of a table are initially activated at program start. If an operation is to be performed on multiple rows, the operation will only be performed on the active rows. The system executive takes into account the status of each row, during table operations, to determine if any action is required. Operations involving only the data locations of a table are not affected by an inactive status. The Language Processor will verify that the Table Name and the Index Name referenced in an ACTIVATE TABLE STATEMENT are properly defined by a Declaration Statement. A Row Designator is a Function Designator used to reference a particular row and therefore is not declared.

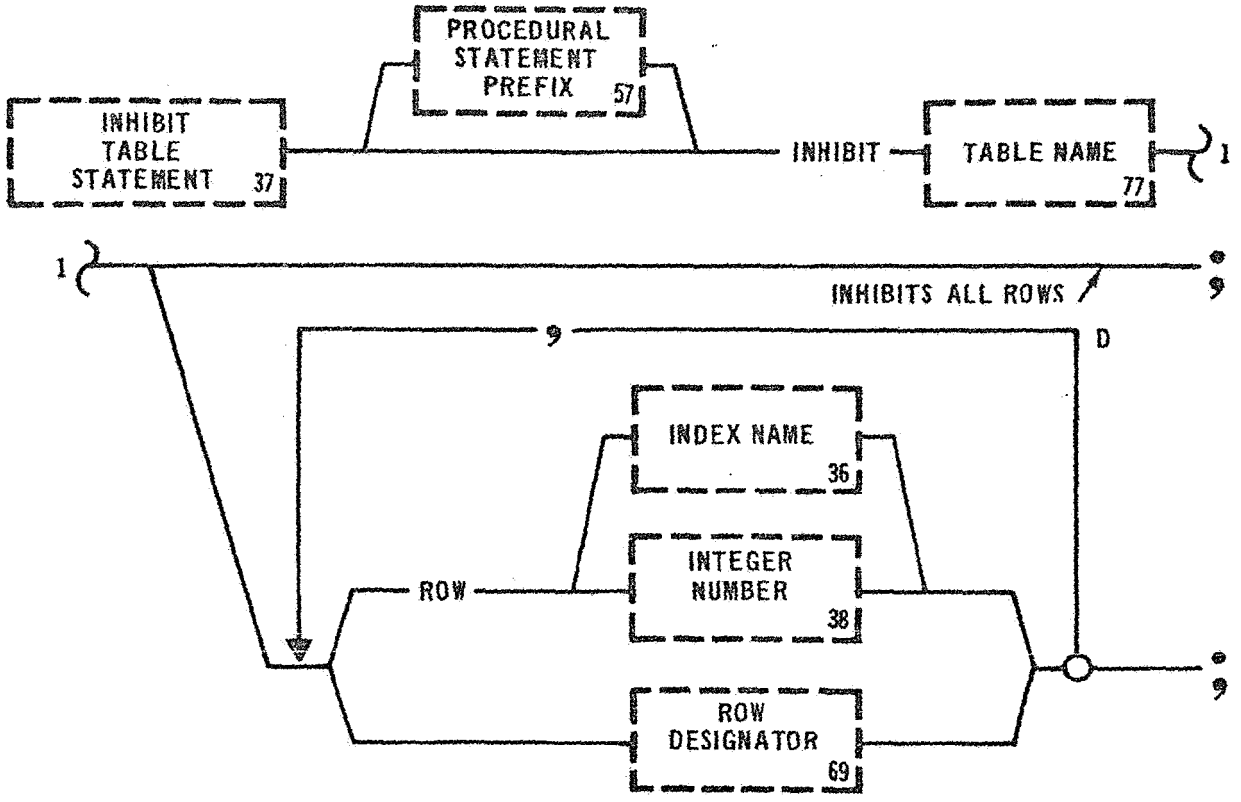
An "ALL" option is provided for activating all rows within a specified table. If a particular row is not referenced the ALL option is assumed.

Examples: ACTIVATE (MESSAGE TABLE) ROW (INDEX);

```
ACTIVATE (POWER ON TABLE)
          <POWER SUPPLY NO 1> ,
          <POWER SUPPLY NO 2> ,
          <MAIN POWER> ;
ACTIVATE (TABLE A) ROW 1, ROW 2, ROW 3;
```

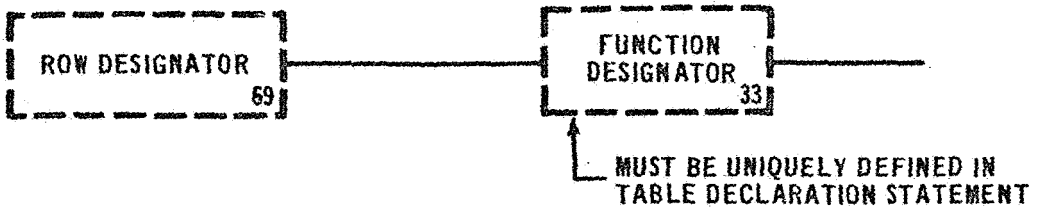
37 REV 0

## INHIBIT TABLE STATEMENT




69 REV 0

## ROW DESIGNATOR



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 3.6.2 Inhibit Table Statement

Examples: INHIBIT (DISCRETE TABLE);

INHIBIT (DISCRETE TABLE) ROW 1;

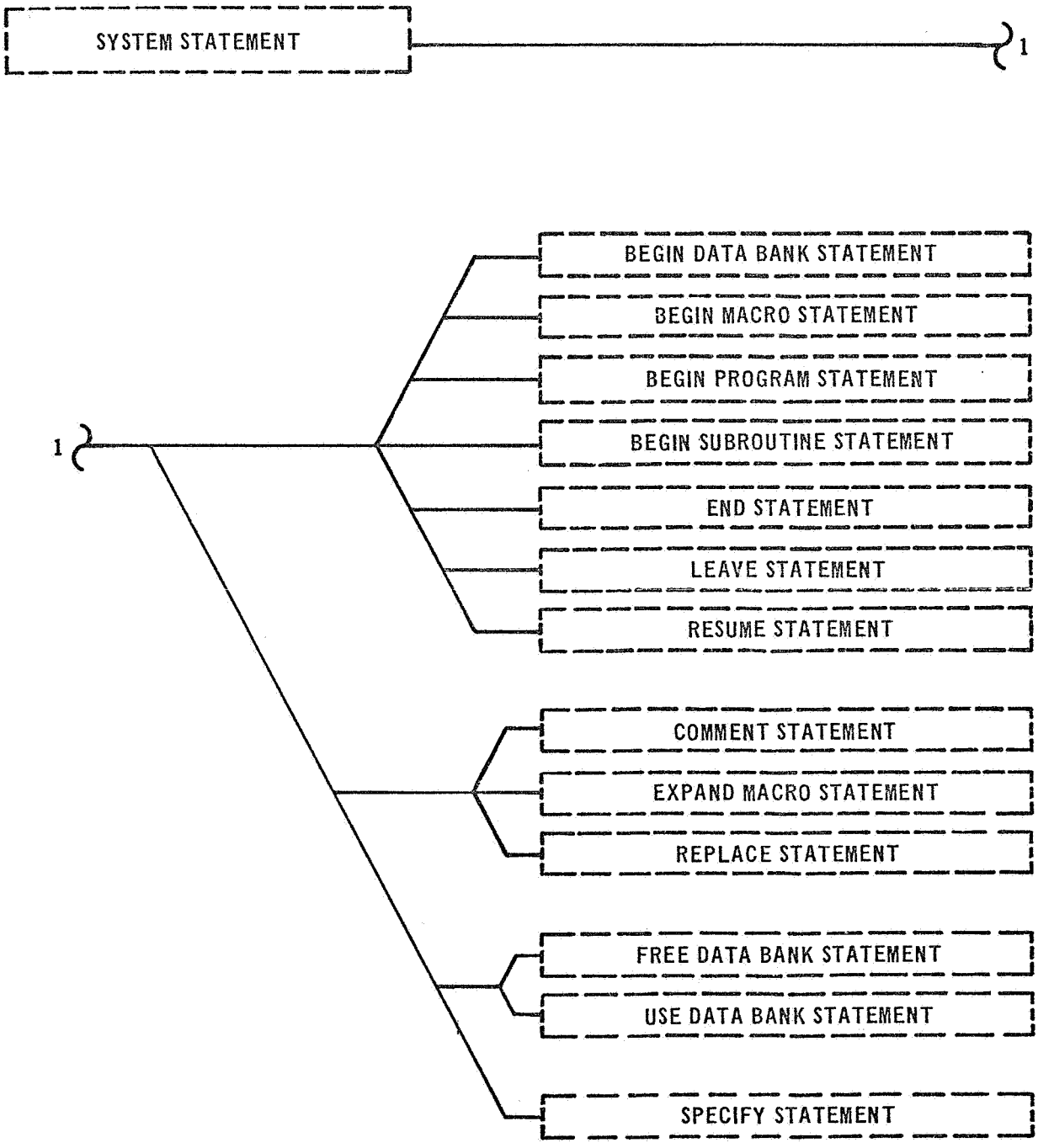
The INHIBIT TABLE STATEMENT assigns a deactive status to a table or individual rows of a table. The INHIBIT TABLE STATEMENT deals only with Function Designators and not with data positions. System under test access via a table cannot be performed on inhibited Function Designators. If an operation is to be performed on a specific row that is inhibited, the operation will not be executed. No error message will occur and the program will then continue normal execution. The Language Processor will verify that the Table Name and Index Names used in an INHIBIT TABLE STATEMENT are properly defined by a Declaration Statement. An Index Name must be declared as an Integer number.

An ALL option is provided for inhibiting all rows within a specified table. The ALL option is invoked if only a Table Name is used.

Examples: INHIBIT (TABLE A) ROW (INDEX NO 1), ROW (INDEX NO 2),
ROW (INDEX NO 3);

INHIBIT (MAIN POWER), <MAIN POWER> , <MAIN POWER> ;

INHIBIT (TABLE A) ROW 1, ROW 2, ROW 3;


SYSTEM STATEMENT
1
1
BEGIN DATA BANK STATEMENT
BEGIN MACRO STATEMENT
BEGIN PROGRAM STATEMENT
BEGIN SUBROUTINE STATEMENT
END STATEMENT
LEAVE STATEMENT
RESUME STATEMENT
COMMENT STATEMENT
EXPAND MACRO STATEMENT
REPLACE STATEMENT
FREE DATA BANK STATEMENT
USE DATA BANK STATEMENT
SPECIFY STATEMENT

# SECTION IV, SYSTEM STATEMENTS

## 4.0 GENERAL

System Statements primarily direct the Language Processor. The System Statements are subdivided into three groups: Boundary Statements, System Directives Statements, and Special Aids Statements.

## 4.1 BOUNDARY STATEMENTS

Boundary Statements delimit a GOAL component. The following statements make up the group of boundary statements:

BEGIN DATA BANK STATEMENT

BEGIN MACRO STATEMENT

BEGIN PROGRAM STATEMENT

BEGIN SUBROUTINE STATEMENT

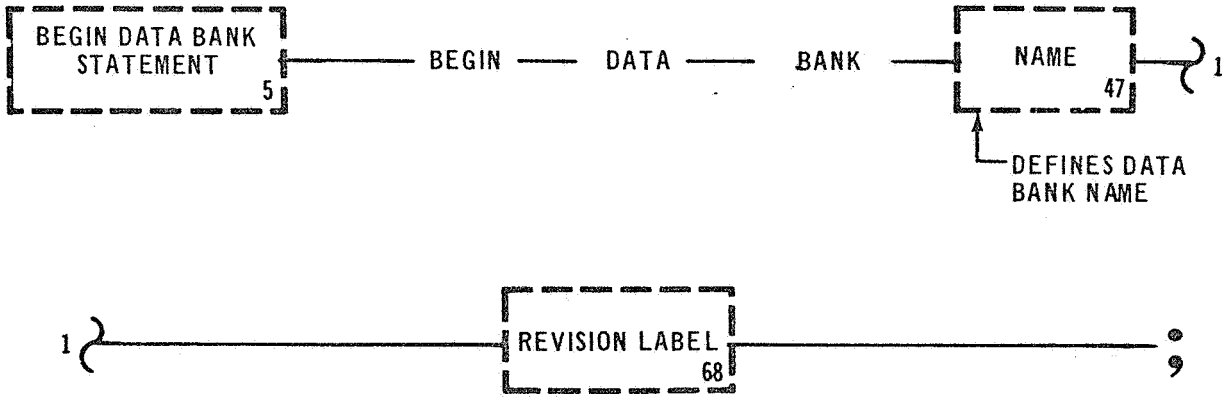
END STATEMENT

LEAVE STATEMENT

RESUME STATEMENT

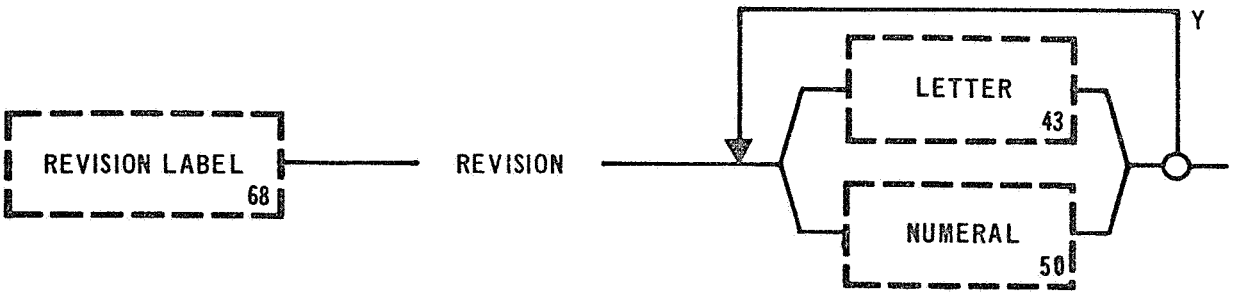
5
REV 0

## BEGIN DATA BANK STATEMENT




68
REV 0

## REVISION LABEL



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

4.1.1 Begin Data Bank Statement

Examples: BEGIN DATA BANK (S2 DATA BANK) REVISION 0;

BEGIN DATA BANK (IU GUIDANCE AND CONTROL) REVISION 1;

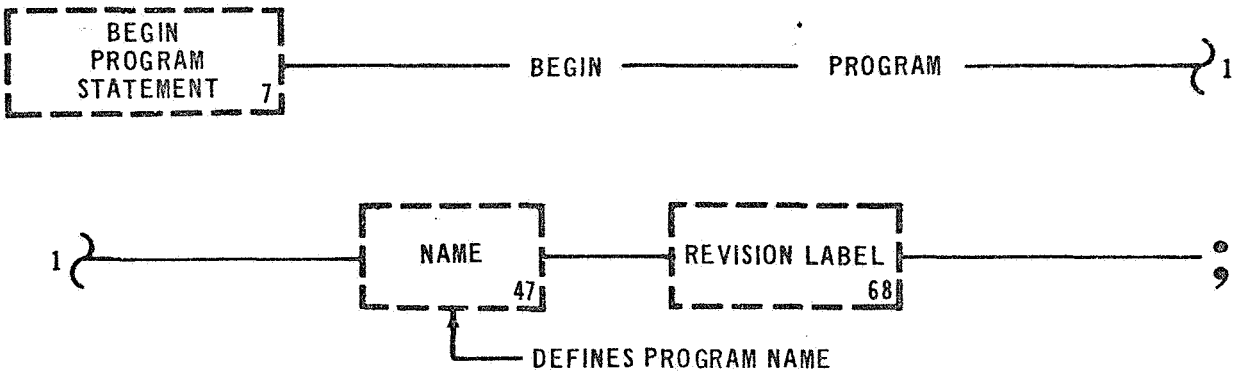
The BEGIN DATA BANK STATEMENT inidcates the start of a Data Bank. This statement is a Language Processor directive that allows a test writer to create an arbitrary name for the identification of a Data Bank. This statement must be the first statement in a Data Bank, and may be used only once in a Data Bank. A Revision Label is required in this statement for proper identification of an individual Data Bank.

Examples: BEGIN DATA BANK (SIC PRESSURE DATA BANK) REVISION 2;

BEGIN DATA BANK (CM DATA BANK) REVISION 10;

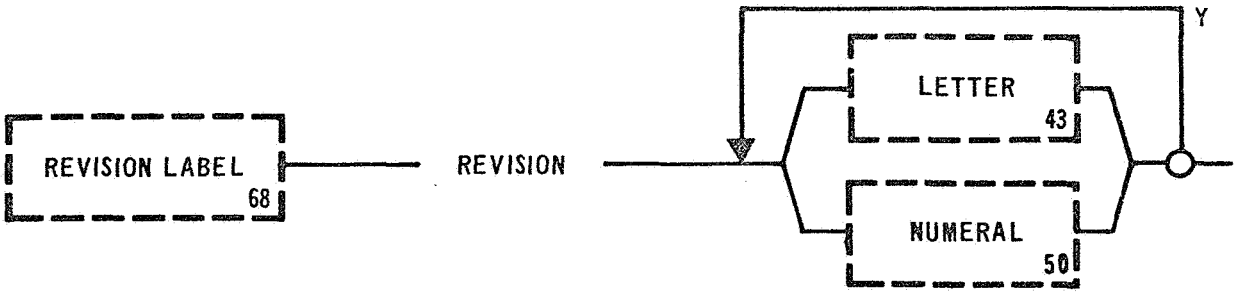
7
REV 0

## BEGIN PROGRAM STATEMENT




68
REV 0

## REVISION LABEL



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 4.1.2 Begin Program Statement

Examples: BEGIN PROGRAM (LV TM CAL) REVISION 0;

BEGIN PROGRAM (CDC SEQ) REVISION 0 COMP 2;

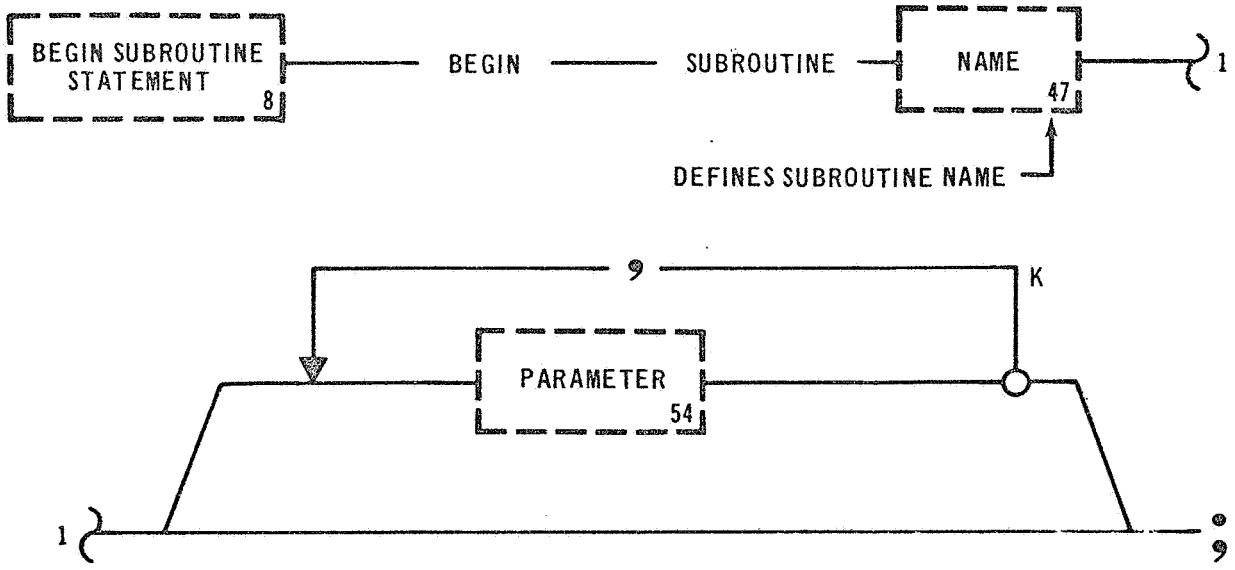
The BEGIN PROGRAM STATEMENT indicates the start of a test program to the Language Processor. This statement allows a test writer to create an arbitrary name for the identification of a test program. This statement must be the first statement in a test program, and may be used only once in a test program. A Revision Label is required in this statement for proper identification of an individual test program.

Examples: BEGIN PROGRAM (PREFLT CAL) REVISION 3;

BEGIN PROGRAM (POWER TRANSFER) REVISION A;

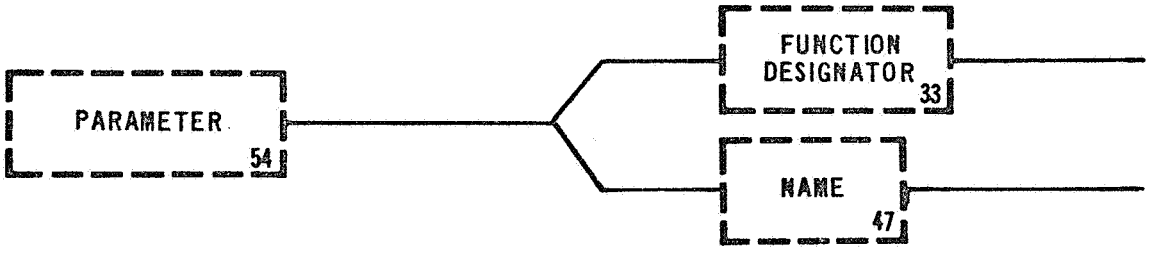
8
REV 0

## BEGIN SUBROUTINE STATEMENT




54
REV 1

## PARAMETER



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 4.1.3 Begin Subroutine Statement

Examples: BEGIN SUBROUTINE (POWER ON);

BEGIN SUBROUTINE (FORCE TERM) (PARAMETER 1);

The BEGIN SUBROUTINE STATEMENT indicates the start of a subroutine. This statement is a Language Processor directive that allows a test writer to create an arbitrary name for the identification of a subroutine. This statement must be the first statement in a subroutine, and may only be used only once in a subroutine. A subroutine may be contained in either a program or data bank. A subroutine may be inserted between any two statements; but for convention, subroutines should be grouped together and inserted after the Declaration Statements. Subroutines also may be compiled as an entry by the Language Processor.

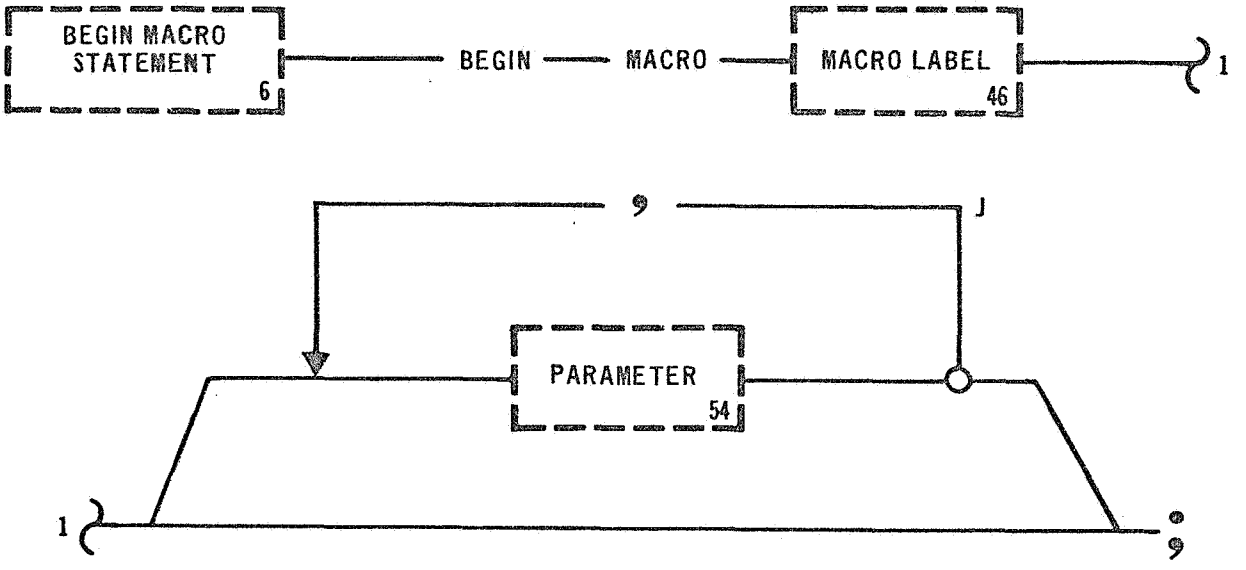
Optional Parameters may be defined with a BEGIN SUBROUTINE STATEMENT. Parameters allow the passing of data from a test program to a subroutine or from a subroutine to a test program. The number of parameters in the BEGIN SUBROUTINE STATEMENT must equal the number of parameters passed by the PERFORM SUBROUTINE STATEMENT.

Examples: BEGIN SUBROUTINE (CALCULATE SIN), (DEGREE);

BEGIN SUBROUTINE (POWER OFF), (GUIDANCE), (CONTROL);

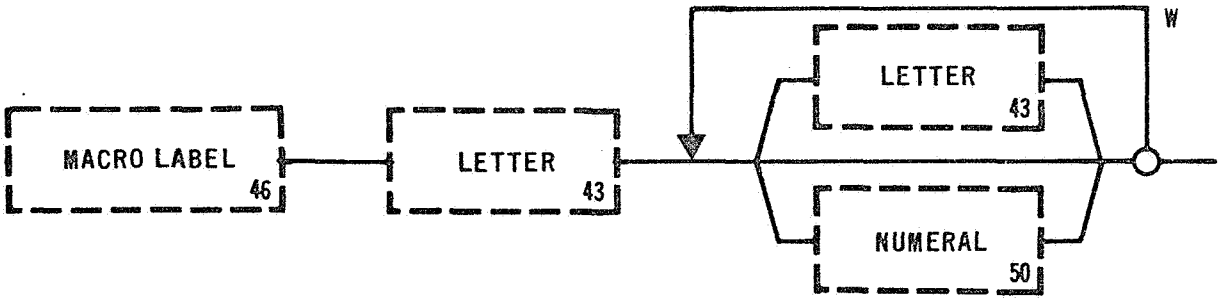
6
REV 0

## BEGIN MACRO STATEMENT




46
REV 0

## MACRO LABEL



### 4.1.4 Begin Macro Statement

Examples: BEGIN MACRO S2 POWER;

BEGIN MACRO A2 (PARAMETER 1);

The BEGIN MACRO STATEMENT indicates the start of a Macro. This statement is a Language Processor directive that allows a test writer to create an arbitrary but unique label for the identification of a Macro. This statement must be the first statement in a Macro and may be used only once in a Macro. A Macro may be contained within either a Program Data Bank or Subroutine. By convention, Macros should be grouped together and inserted immediately following the BEGIN PROGRAM STATEMENT of the host component.

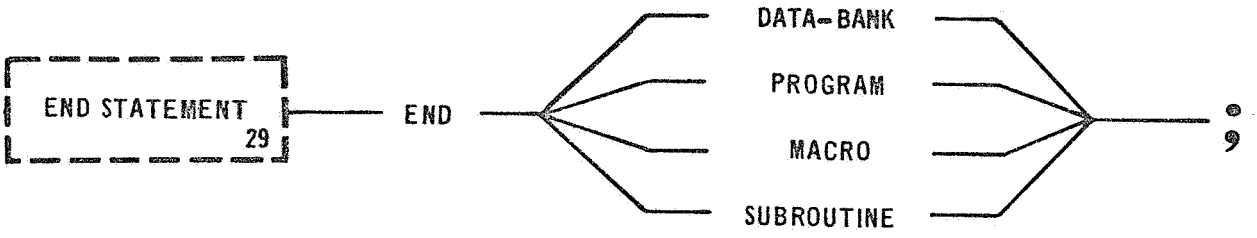
Optional Parameters may also be defined with a BEGIN MACRO STATEMENT. These Parameters are used to indicate substitutable arguments within a Macro skeleton. The substitutable arguments are replaced by the corresponding Character Strings defined in the EXPAND MACRO STATEMENT. This replacement is done during program completion by the Language Processor. For more information on Macros refer to Section V.

Examples: BEGIN MACRO ADJUST (FUNCTION DESIGNATOR), (VOLTAGE ADJUST),
(STEP);

BEGIN MACRO CALCULATE SINX (RADIAN);

29
REV 0

## END STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 4.1.5 End Statement

Examples: END DATA BANK;

END PROGRAM;

The END STATEMENT indicates the end of a referenced component. The END STATEMENT has four options: Data Bank, Program, Macro, and Subroutine. The Language Processor will verify that the referenced package is consistent with the package in which the END STATEMENT is contained.

The END SUBROUTINE option, when encountered during execution, will cause control to be returned to the calling program.

The END PROGRAM option indicates that the execution of a test program is complete. It also indicates to the Language Processor the end of a program compilation.

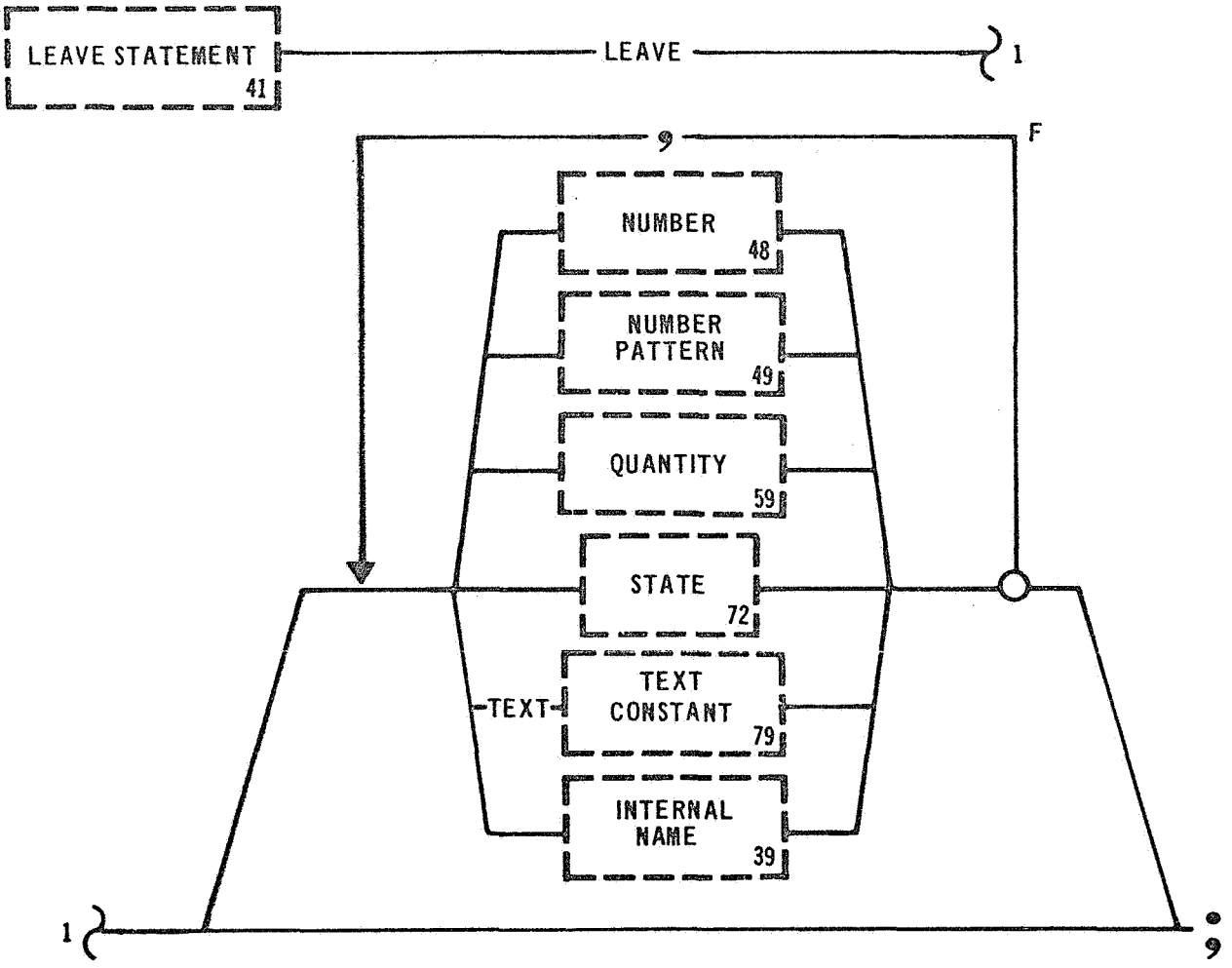
The END DATA BANK, and END MACRO options serve only as an indicator to the Language Processor. If the statement preceding the END STATEMENT is not a TERMINATE STATEMENT, then the END STATEMENT also defaults to produce the effect of a TERMINATE STATEMENT.

Examples: END MACRO;

END SUBROUTINE;

41
REV 1

## LEAVE STATEMENT



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 4.1.6 Leave Statement

Examples: LEAVE;

LEAVE 10,10,20;

The LEAVE STATEMENT is a Language Processor directive providing a method for leaving GOAL during processing of a Data Bank Subroutine and enabling the execution of a subroutine written in another language. It also enables the passing of data to the other language subroutine. The other language must be compatible with the GOAL programming system before its subroutine capabilities can be used. The LEAVE STATEMENT is a legal statement only in a Data Bank Subroutine.

To use a subroutine written in another language the subroutine must first be compiled by the other language compiler. The explicit error free object subroutine is then inserted within the Data Bank Subroutine.

The LEAVE STATEMENT must immediately precede the other language subroutine. The LEAVE and RESUME statements mark the beginning and ending, respectively, of the other language subroutine. The PERFORM SUBROUTINE STATEMENT is used to execute a subroutine containing a NON-GOAL component. The Subroutine Name used by the PERFORM SUBROUTINE STATEMENT references a Data Bank Subroutine that contains the NON-GOAL component which is to be executed.

Examples: LEAVE TEXT (DATE 10/28/72), 14.60 PSIA, 14.65 PISA, 28 MPH;

LEAVE (VARIABLE DATA), ON, OFF, X10AB, X10AC;

67
REV 1

## RESUME STATEMENT

| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 4.1.7 Resume Statement

Example: RESUME;

The RESUME STATEMENT is a Language Processor directive. It provides a method of resuming the compilation of a GOAL subroutine. It also enables a return to the host Data Bank Subroutine after the execution of a NON-GOAL subroutine. The RESUME STATEMENT must immediately follow the NON-GOAL component. A GOAL statement must follow the RESUME STATEMENT.

## 4.2 SYSTEM DIRECTIVES

The System Directive Statements relate a test program to a specified Data Bank. The USE DATA BANK STATEMENT and FREE DATA BANK STATEMENT are directives to the Language Processor. The SPECIFY STATEMENT is used in creating a Data Bank. It relates a Function Designator to a test point. The System Directive Statements are:

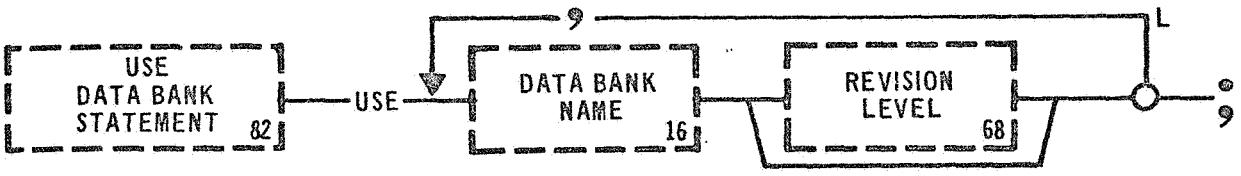
USE DATA BANK STATEMENT

FREE DATA BANK STATEMENT

SPECIFY STATEMENT

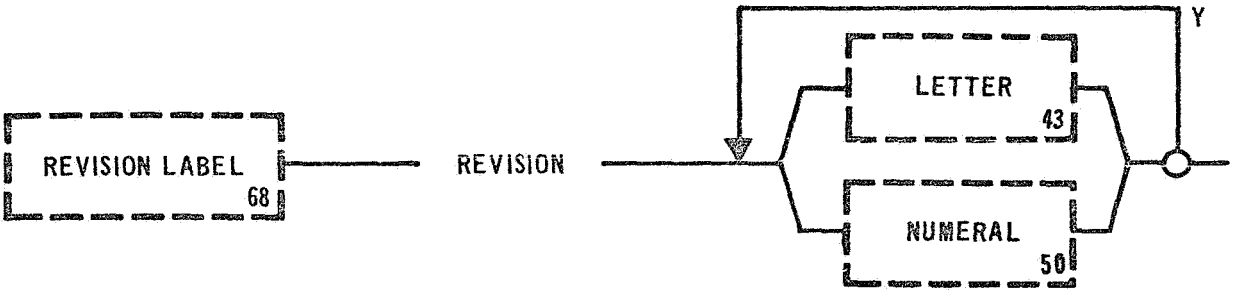
82 REV 0

## USE DATA BANK STATEMENT




68 REV 0

## REVISION LABEL



### 4.2.1 Use Data Bank Statement

Examples: USE (S2 DATA BANK), (SIC DATA BANK), (SIVB DATA BANK),
(IU DATA BANK);

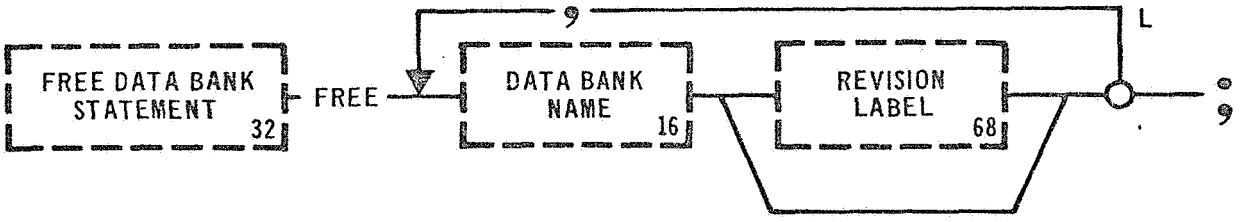
The USE DATA BANK STATEMENT is a Language Processor directive used to identify a unique Data Bank for compilation of a test program or part of a test program. This statement allows the Language Processor to have access to a referenced Data Bank. More than one Data Bank may be active at a time. However, an increase in the number of active Data Banks may result in an increase in time required for compilation. The activation of an additional Data Bank will not effect the status of Data Banks that have been previously activated.

The Revision Label is optional, and if not used, the first Data Bank located by the system that contains a matching name will be used regardless of its-Revision Label.

The Language Processor will flag an error condition if a name is used to reference a nonexisting Data Bank or a Data Bank that has not been processed.

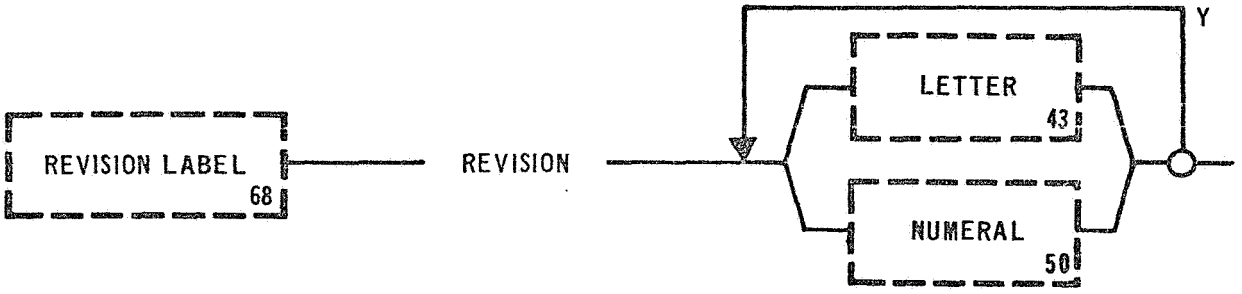
32
REV 0

## FREE DATA BANK STATEMENT




68
REV 0

## REVISION LABEL



### 4.2.2 Free Data Bank Statement

Examples: FREE (SIC DATA BANK);

FREE (IU DATA BANK) REVISION 2;

The FREE DATA BANK STATEMENT is a Language Processor directive. It inhibits the Language Processor from accessing a previously activated Data Bank. To release a Data Bank it must have been previously activated by a USE DATA BANK STATEMENT. Any attempt to free a non-active Data Bank will result in a processor error.

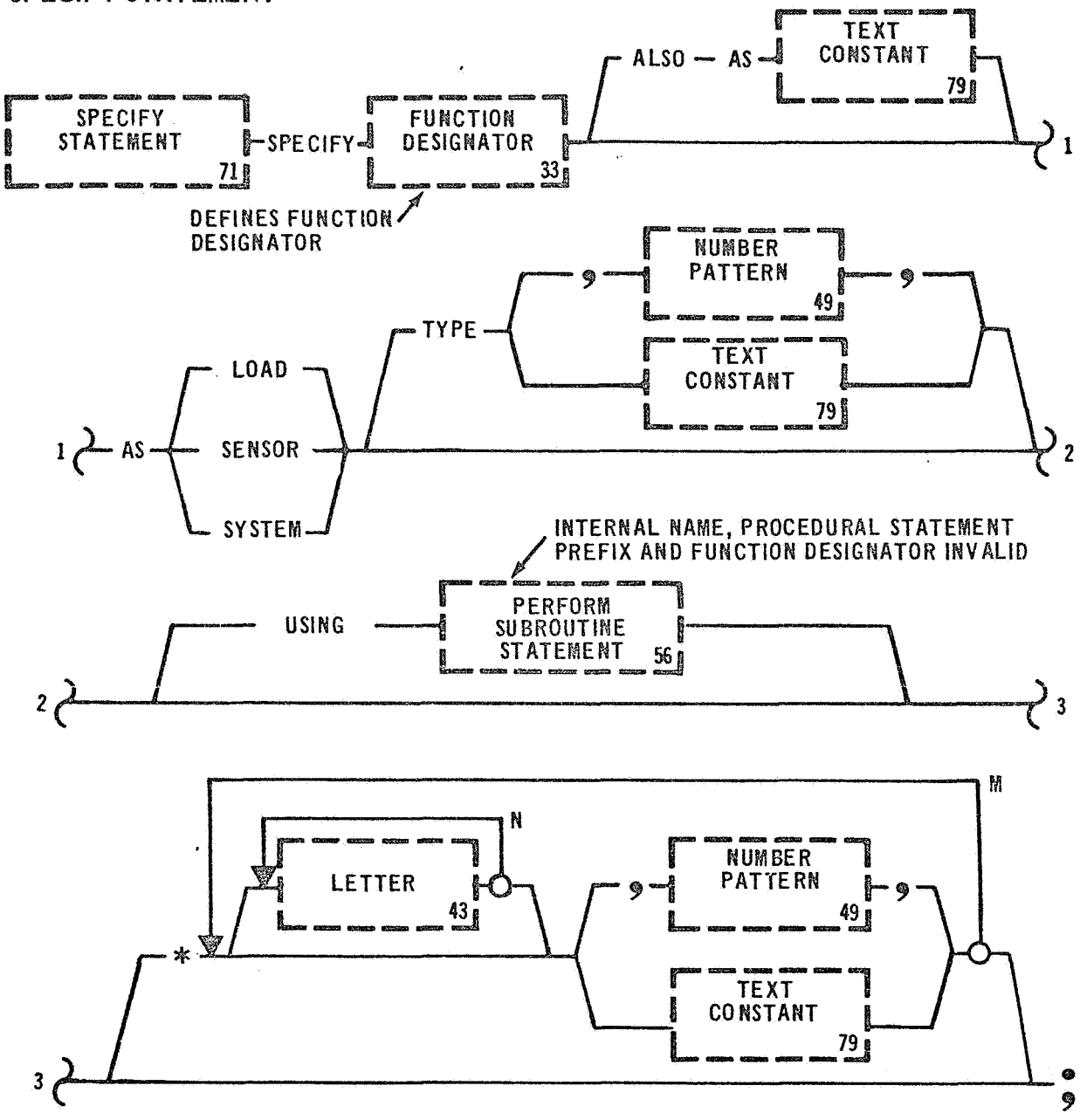
The Revision Label is optional and may be used to specify a specific update level of a Data Bank. If two Data Banks have the same name but different Revision Labels, and the Revision Label is not specified in a FREE DATA BANK STATEMENT, the first Data Bank found by the Language Processor will be released.

Examples: FREE (SIC DATA BANK), (S2 DATA BANK);

FREE (S2 MEC DATA BANK) REVISION 1, (S2 ELEC DATA BANK)
REVISION 2, (IU ELEC DATA BANK);

71
REV 0

## SPECIFY STATEMENT



point for concurrent testing. Synchronization points in two or more concurrently executing programs can cause the tests to be synchronized at that point. The Language Processor will verify the relationship between a Function Designator and the statement in which it is used. For example, a Function Designator declared as a Load and used in a READ STATEMENT is considered illegal. The three main forms of Function Designators may be subdivided, if necessary, as to the type of Function Designator.

An optional subroutine that provides the necessary code conversation is specified for any inputs or outputs. The last optional branch started with an asterisk symbol is used to further define a Function Designator. For example in the case of telemetry, the channel address may be defined.

```
Examples:  SPECIFY<OTBD CTS> ALSO AS (OUTBOARD CUTOFF SIGNAL)
                SENSOR TYPE (DDAS) * ADDRESS (AP1A0-14-00-00);
           SPECIFY<CDF> ALSO AS (COUNTDOWN FLAG) SYSTEM TYPE
                (STATE)  * (ON);
```

## 4.3 SPECIAL AID STATEMENTS

The Special Aid Statements ease the burden on the test writer and reviewer. These statements are Language Processor directives and are not executable. The REPLACE STATEMENT and EXPAND MACRO STATEMENT are used for substituting source code during processing. The COMMENT STATEMENT may contain any commentary desired by the test writer. The contents of the statement are printed but are otherwise ignored by the Language Processor.

The Special Aid Statements are:

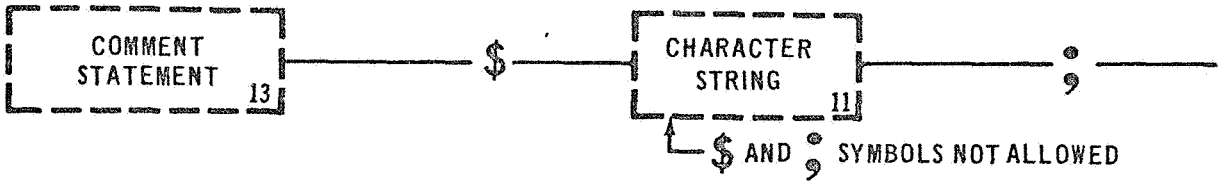
COMMENT STATEMENT

EXPAND MACRO STATEMENT

REPLACE STATEMENT

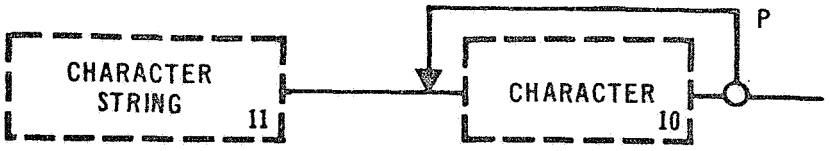
13
REV 1

## COMMENT STATEMENT




11
REV 0

## CHARACTER STRING



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 4.3.1 Comment Statement

Examples: $ POWER TRANSFER SWITCH VERIFICATION;

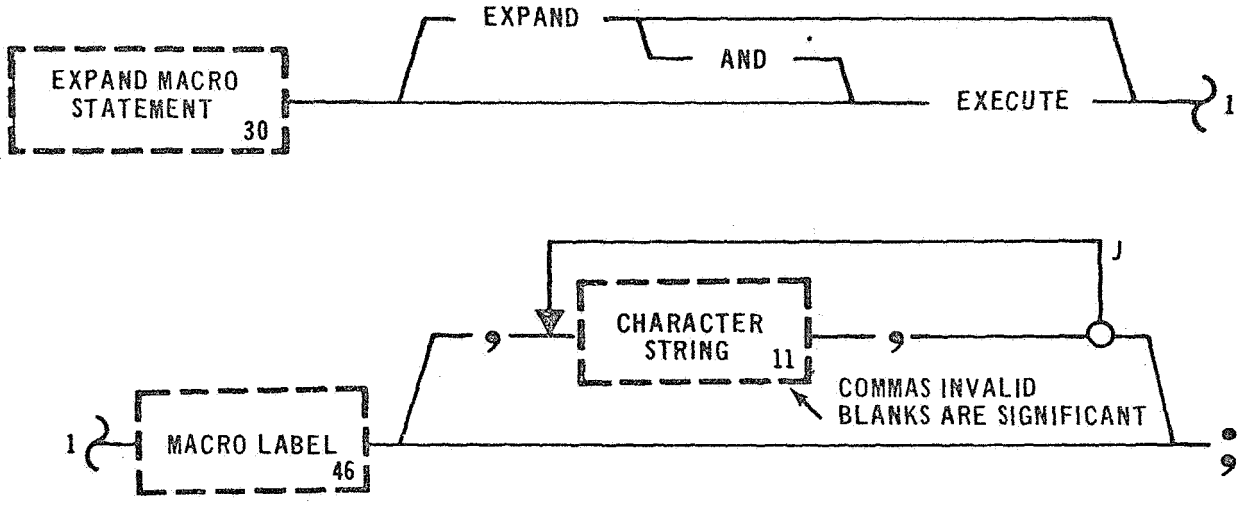
$ SCAN ERROR ROUTINE;

$ 15;

All comments must begin with a currency symbol and terminate with a semicolon. Comments may be inserted at any point in a program except where specifically prohibited by syntax notes. Comments primarily benefit the program reviewers and have no effect on the execution of the program. They will be printed with the compile listings but will be otherwise ignored. Currency symbols and semicolons are not allowed in the body of the comment.

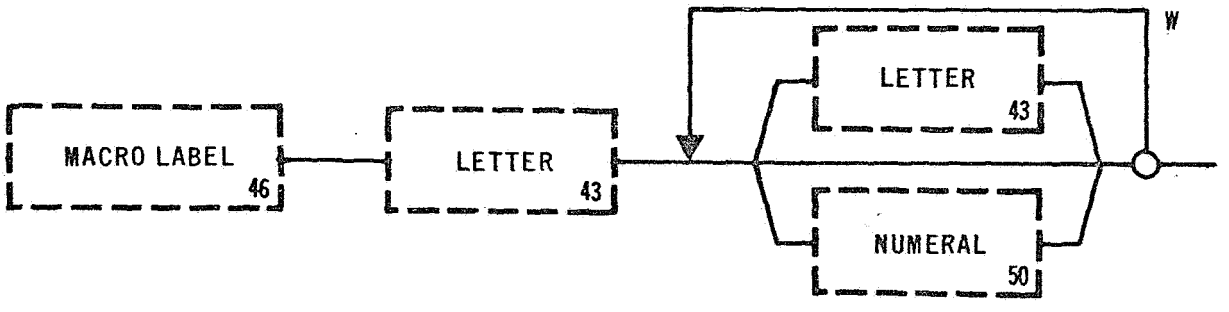
30 REV 0

## EXPAND MACRO STATEMENT




46 REV 0

## MACRO LABEL



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

4.3.2 Expand Macro Statement

Examples: EXPAND MACRO CALCULATE SINX, (X),;

EXECUTE MACRO ADJUST, < AC SIGNAL > , (0.5V), S340,;

EXPAND AND EXECUTE MACRO REMOVE POWER, (CAL TABLE), 3,;

The EXPAND MACRO STATEMENT causes the Macro skeleton, a predefined sequence of source code, to be inserted by the Language Processor into the source program at the point occupied by the EXPAND MACRO STATEMENT. Once a Macro is defined it may be expanded at all component levels. The EXPAND MACRO STATEMENT provides a means for the substitution of parameters, as defined by the BEGIN MACRO STATEMENT, into the source code created by the test writer.

The EXPAND MACRO STATEMENT has three primary options that control only the printing of the Macro skeleton. The three options are the EXPAND, EXECUTE, and the EXPAND AND EXECUTE options. An example of a Macro might be:

```
BEGIN MACRO ADJUST (UNIT), (INCREMENT), (STEP 2);
LET (VOLTS) = (0.5V);
(STEP 2) APPLY (VOLTS) TO (UNIT);
LET (VOLTS) = (VOLTS) + (INCREMENT);
```

```
DELAY 2 SECS;
VERIFY (UNIT) IS LESS THAN 28V THEN GO TO (STEP 2);
END MACRO;
```

The above sequence of source codes, statements, is defined as the Macro skeleton.

The EXPAND option inhibits the printing of the EXPAND MACRO STATEMENT, and allows for the printing of the Macro skeleton after the Parameters have been substituted. Using the previously defined Macro, an example of the EXPAND option might be:

```
EXPAND MACRO ADJUST,<AC SIGNAL> , (0.2V), S495,;
```

This statement would result in the following printout:

```
LET (VOLTS) = (0.5V);
STEP 495 APPLY (VOLTS) TO<AC SIGNAL> ;
LET (VOLTS) = (VOLTS) + (0.2V);
DELAY 2 SECS;
VERIFY<AC SIGNAL> IS LESS THAN 28V THEN GO TO STEP 495;
```

The EXECUTE option allows for a partial printing of the EXPAND MACRO STATEMENT. This option inhibits the printing of the Macro skeleton with the substituted Character Strings. Using the above defined Macro, an example of the EXECUTE option might be:

```
EXECUTE MACRO ADJUST,<AC SIGNAL> , (0.1V), S540,;
```

This statement would result in the following statement to be printed:

```
ADJUST,<AC SIGNAL> , (0.1V), S450,;
```

The Macro-generated statements are inserted but are not printed in this option.

The EXPAND AND EXECUTE option is a combination of the two other options. For example, given the EXPAND MACRO STATEMENT:

EXPAND AND EXECUTE MACRO ADJUST, < AC SIGNAL > , (0.3V), S540,;

would result in the following statements to be printed in a source listing:

```
ADJUST < AC SIGNAL > , (0.3V), S540,;
LET (VOLTS) = (0.5V);
STEP 540 APPLY (VOLTS) TO<AC SIGNAL> ;
LET (VOLTS) = (VOLTS) + (0.3V);
DELAY 2 SECS;
VERIFY<AC SIGNAL> IS LESS THAN 28V THEN GO TO STEP 540;
```

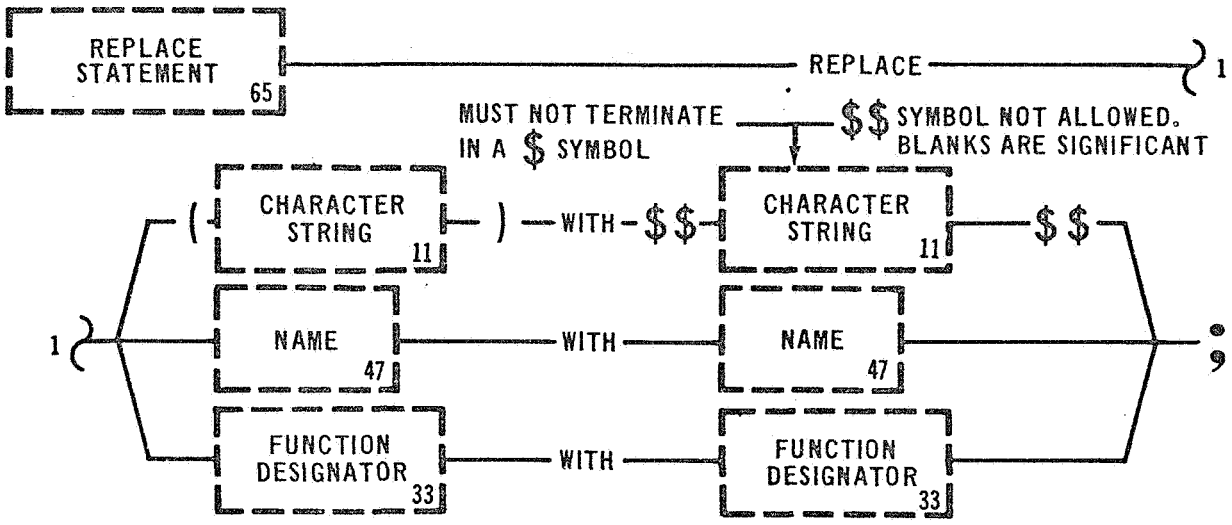
Examples: EXPAND MACRO APPLY S 2 POWER;

EXECUTE MACRO CALCULATE SINX, 3 VOLTS,;

EXPAND AND EXECUTE MACRO X,< SWITCH 3 > , AND SAVE AS,;

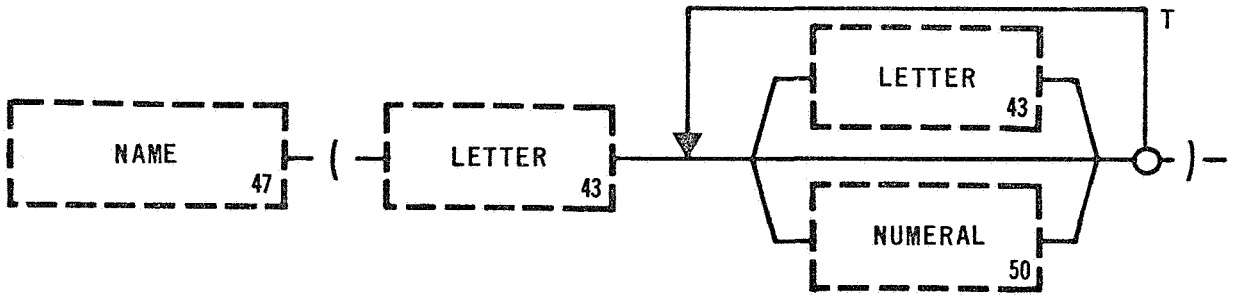
65 REV 0

## REPLACE STATEMENT




47 REV 0

## NAME



| DECLARATION |
| --- |
| PROCEDURAL |
| SYSTEM |

### 4.3.3 Replace Statement

Examples: REPLACE < POWER SUPPLY NO 1 > WITH < POWER SUPPLY NO 2 > ;

REPLACE (DISCRETE TABLE) WITH (DISCRETE TABLE LDI);

The REPLACE STATEMENT is a Language Processor directive. It provides a means for substituting source code during processing. This capability allows a convenient means for updating Function Designators and Internal Names. It may also be used as a form of shorthand to ease the burden for a test writer. The source code on the right side of a REPLACE STATEMENT will be substituted for the source code identified by the left side of the statement. The Language Processor will compare all names and Function Designators, within a test procedure, to the source code identified on the left side of a REPLACE STATEMENT. If a comparison is made, the source code on the right side of the REPLACE STATEMENT will replace the old source code within the procedure. Once a REPLACE STATEMENT is defined it may be used to substitute at all component levels. A test program listing generated by the Language Processor will reflect any substitution caused by a REPLACE STATEMENT.

The double currency symbols used when replacing a Character String are dropped off by the Language Processor, and only the source code between the double currency symbols is substituted. The parenthesis and angle brackets are substituted.

Examples: REPLACE (A) WITH $$AND SAVE AS$$

REPLACE (B) WITH $$AND SAVE AS (RESERVE)$$;

# SYSTEM V, SYSTEM CONCEPTS

## 5.0 GENERAL

This section discusses some of the general concepts used in developing the GOAL programming system. The system is divided into three main areas: (1) the language, (2) the Language Processor and (3) the System Executive.

The GOAL Language consists of five components which are defined as: (1) Program, (2) Data Bank, (3) Subroutine, (4) Macro, and (5) Non-GOAL. Certain combinations and constraints of these components are discussed in this section along with various interactions between individual components.

Components are a necessary ingredient in any programming system, but they do not make up a complete system. These components need support; this type of support comes in the form of a processor and an executive. A GOAL processor and executive are also discussed in this section.

Finally, this section discusses some general programming concepts in the following areas: concurrency, tables, interrupts, terminations, and dimensions.

## 5.1 ALLOWABLE STRUCTURES

When visualizing automatic processing of procedures, several distinct components of the system emerge. GOAL was designed to be compatible with the following language system concepts. "Test functions" will be contained primarily in the "program" and "subroutine" components and since GOAL was developed for "test functions," GOAL appears most natural in those components.

The two primary (stand-alone) components are:

1. Program
2. Data Bank

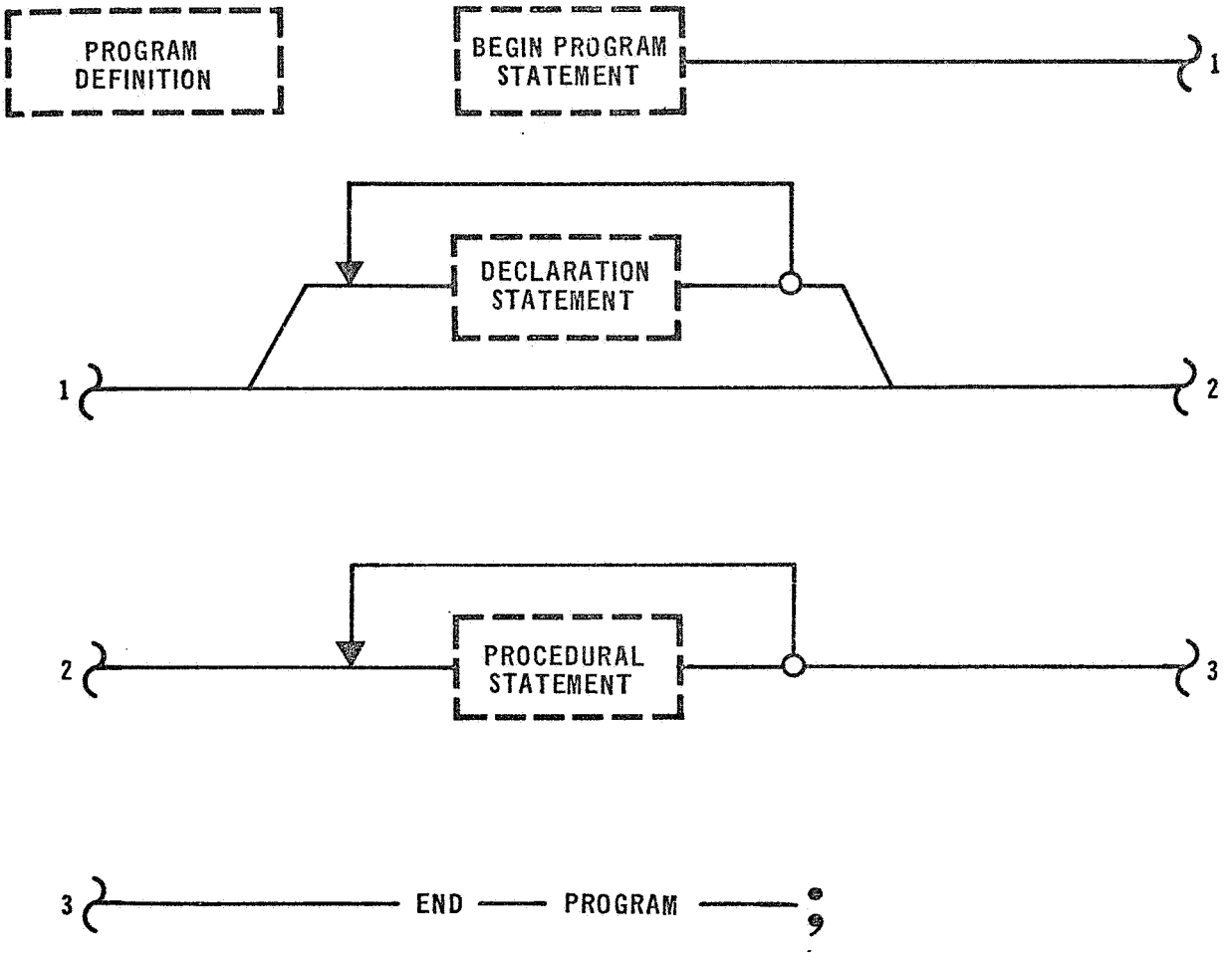
The three secondary components are:

1. Subroutine
2. Macro
3. Non-GOAL

A Subroutine must be contained either within a Program or Data Bank. Subroutines in a Program should, by convention, be grouped together and inserted following the Declaration Statements, if any, and prior to the Procedural Statements. However, Subroutines may be inserted between any two statements of a Program or Data Bank.

A Macro may be contained within either a Program, Data Bank, or a Subroutine. Macro's must be defined prior to the statement that uses (expands) it. By convention, Macros should be grouped together and inserted immediately following the Begin Statement of the host component. Once inserted, the Macro may be expanded at all component levels (i.e., global).

A Non-GOAL component must be contained within a Subroutine and that Subroutine must be within a Data Bank. Non-GOAL components may be inserted at any point within a Data Bank Subroutine provided it does not divide or break a statement. This allows a Data Bank Subroutine to consist solely of a Non-GOAL component if so desired.


PROGRAM DEFINITION
BEGIN PROGRAM STATEMENT
1
DECLARATION STATEMENT
1
2
PROCEDURAL STATEMENT
2
3
3
END
PROGRAM
;

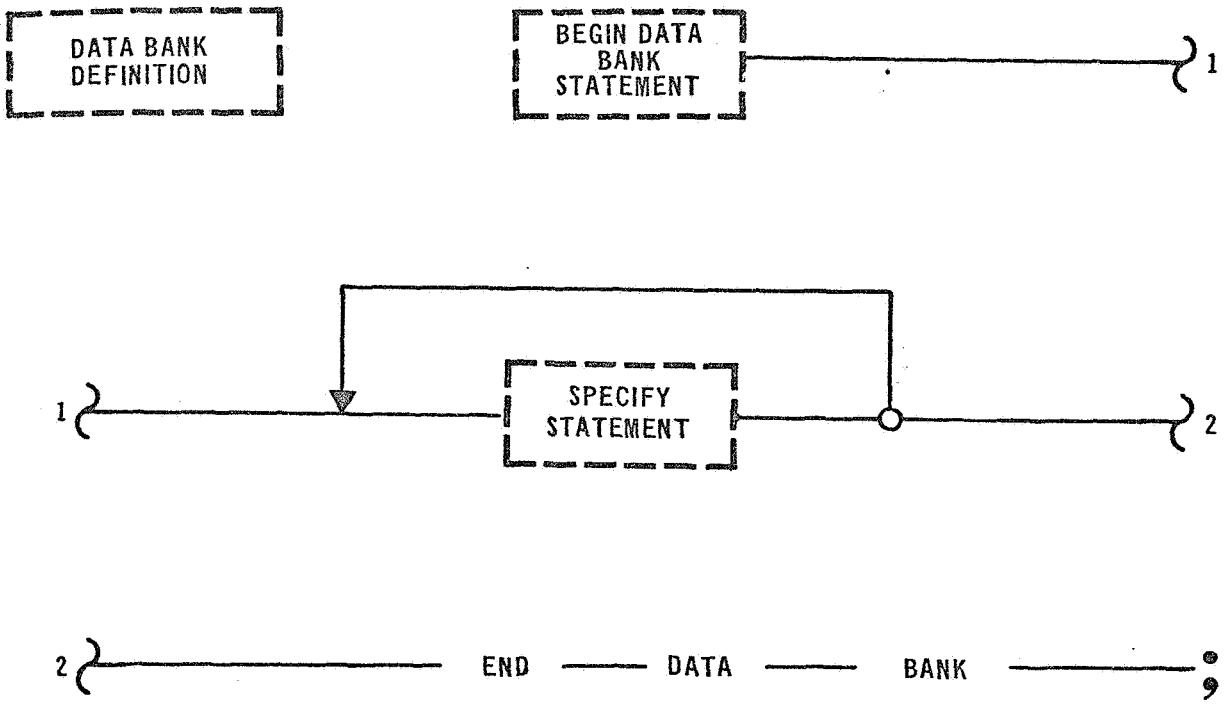
### 5.1.1 Program

A Program is the highest component available in the language. A Program must start with a BEGIN PROGRAM STATEMENT and end with an END STATEMENT. A Program has two main parts: (1) a Declaration Section, and (2) a Procedural Section.

The Declaration Section consists of Declaration Statements. This section is optional. All Names used within the Procedural Section must be declared; therefore, the use of the Declaration Section depends upon the use of Names. If no Names are used in the Procedural Section, the Declaration Section is not required. Data defined in the Declaration Section is valid only in that Program.

A Procedural Section is required. It consists of Procedural Statements which do the actual testing of the system under test. A Program may contain only one BEGIN PROGRAM STATEMENT. It may also contain only one END STATEMENT which invokes the End Program option.

All Statement Numbers are local to the program; that is, a Procedural Statement cannot reference a Statement Number located within a Subroutine. Likewise all Names are local.


DATA BANK DEFINITION
BEGIN DATA BANK STATEMENT
1
SPECIFY STATEMENT
1
2
2
END
DATA
BANK
;

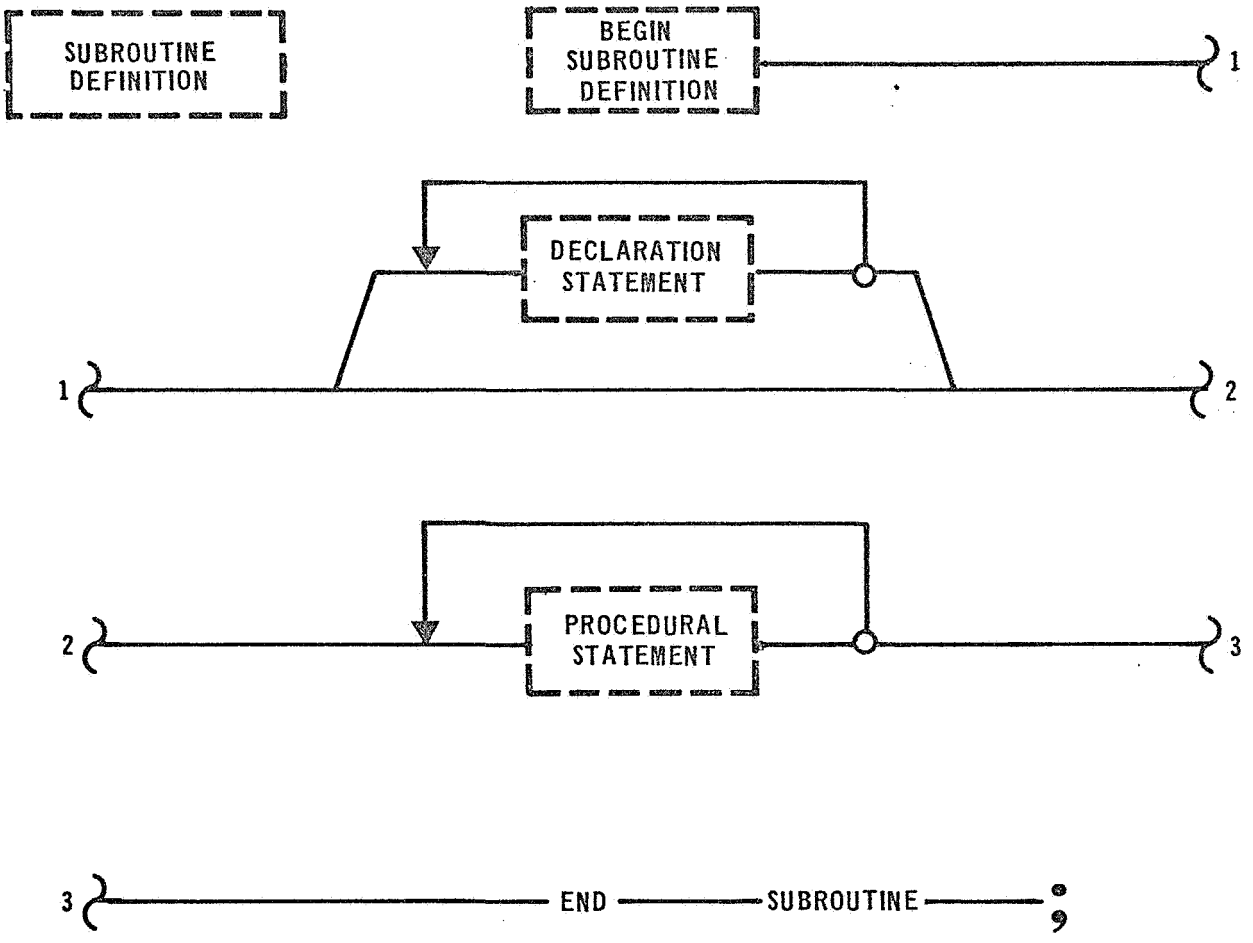
### 5.1.2 Data Bank

The Data Bank must be bounded by a BEGIN DATA BANK STATEMENT and an END DATA BANK STATEMENT. The only statements allowed in the Data Bank are the SPECIFY STATEMENTS. This should not be confused with the option to insert the other GOAL components, specifically Subroutines and Macros, between any two SPECIFY STATEMENTS. The informational content of the Data Bank indicates a data set with the following characteristics:

1. The Data Bank must be capable of storing unique data for a large number of individual entries.
2. The ability to reference, via alphanumeric name, any entry in the Data Bank.
3. Most of the content of the Data Bank will be entries related to Function Designators but Subroutines will also be present.

The Data Bank will be present only during compile time. The actual Data Bank entries are directly dependent upon hardware design.

The Data Bank provides the necessary linkage between Function Designators and the System Under Test. The use of particular Data Banks can be controlled during compilation by the USE DATA BANK STATEMENT and the FREE DATA BANK STATEMENT.


SUBROUTINE DEFINITION
BEGIN SUBROUTINE DEFINITION
1
DECLARATION STATEMENT
1
2
PROCEDURAL STATEMENT
2
3
3
END
SUBROUTINE
;

### 5.1.3 Subroutines

A Subroutine is bounded by a BEGIN SUBROUTINE STATEMENT and an END STATEMENT. It may have a Declaration Section, but must have a Procedural Section. Parameters may be passed to the Subroutine through the BEGIN SUBROUTINE STATEMENT. As in the Program, the use of a Declaration Section depends upon the use of Names in its Subroutine since all Names must be declared. All Names declared are local; i.e., Names declared in a Subroutine cannot be referenced by the calling program.

#### 5.1.3.1 Program Subroutine Interaction

The Subroutine may be executed only via a PERFORM SUBROUTINE STATEMENT. This statement will carry the Parameters, if any, to be passed to the Subroutine. If this statement also carries the adjective, CRITICAL, then the execution of the Subroutine will not be broken by a Language Level Interrupt occurrence.

#### 5.1.3.2 Program Subroutine Similarities

The Program component and Subroutine component are similar in many ways:

1. Both may be entered only through their Begin Statements.
2. All branching is local; i.e., once inside a Subroutine, as with a Program, you cannot branch outside its boundaries, and you cannot branch into a Subroutine or Program from an external source.
3. Neither allow component nesting; i.e., no Begin Statement of the same component is allowed between the Boundary Statements of the component.

4. All Procedural and Declaration Statements are legal.
5. Data defined in a Program is local to that Program and data defined in a Subroutine is local to that Subroutine. If a Name is to be used in a Subroutine, it must be declared in the Subroutine.
6. Both may be terminated by a TERMINATE STATEMENT or by a termination default when an END STATEMENT is encountered.

Note: A Program is not terminated when a Subroutine encounters a terminate command, unless it is a terminate system command.

Execution is returned to the next Program statement after the calling PERFORM SUBROUTINE STATEMENT. However, if the execution of the Subroutine is due to a Language Level interrupt, then control is returned to Program as directed by the WHEN INTERRUPT STATEMENT.

5.1.3.3 Program Subroutine Differences

There are several ways in which Subroutine components differ from Program components. These are:

1. A Subroutine may be executed only via a PERFORM SUBROUTINE STATEMENT.
2. A Subroutine inserted in a Program is available only to that Program. By convention it is recommended that all Subroutines be placed between the Declaration and Procedural Sections of the Program component. This is not a requirement; however, and Subroutines may actually be placed between any two statements either in a Program or a Data Bank.
3. Data defined outside a Subroutine is available only through the Begin Statement.

5.1.3.4 Subroutine Advantages

It is to the procedure writer's advantage to use Subroutines whenever a task is to be performed more than once in his procedure because:

1. It provides more efficient coding.
2. It provides procedure modularity resulting in ease of change and debug.
3. It provides quicker and easier review of a procedure by allowing a repeatable task to be performed and then return to the next statement following the PERFORM SUBROUTINE STATEMENT in the Program. For example: in the following Subroutine, control will be returned to Step 2 after the execution of the END STATEMENT in the Subroutine.

5.1.3.5 Subroutine Example

| GOAL PROGRAM | GOAL SUBROUTINE |
|---|---|
| _____ | BEGIN SUBROUTINE (STATUS CHECK) |
| _____ | <A> ,<B> ; |
| OPEN <LOX VENT VALVE> ; | VERIFY <A> IS ON; |
| PERFORM SUBROUTINE (STATUS CHECK) | VERIFY <B> IS ON; |
| <BUS 1> ,<BUS 2> ; | TURN ON <SYSTEM INDICATOR> ; |
| STEP 2 CLOSE <LOX VENT VALVE> ; | END SUBROUTINE; |
| _____ | |
| _____ | |
| PERFORM SUBROUTINE (STATUS CHECK) | |
| <BUS 3> ,<BUS 4> ; | |
| _____ | |
| _____ | |

_____Implies one or more statements

5.1.3.6 Subroutine Parameter Replacement

| PROGRAM | SUBROUTINE |
|---|---|
| DECLARE QUANTITY (Z) | BEGIN SUBROUTINE (SA) |
| ——— | <A>, (B), (C), (E); |
| ——— | DECLARE QUANTITY (B); |
| ——— | DECLARE NUMBER (D); |
| PERFORM SUBROUTINE (SA) | |
| <CONSOLE 1>, (Z), 15V, 2; | TURN ON<SWITCH 1> ; |
| ——— | DISPLAY TEXT (BEGIN SA) TO<A> ; |
| ——— | READ<BUS 1> AND SAVE AS (B); |
| ——— | APPLY (C) TO<BUS 2> ; |
| | LET (D) = 3 + (E); |
| | IF (D) = 7 THEN TERMINATE; |
| | TURN OFF<SWITCH 1> ; |
| | END SUBROUTINE; |

After execution of Subroutine (SA) the parameters will be replaced as follows:

TURN ON<SWITCH 1> ;

DISPLAY TEXT (BEGIN SA) TO<CONSOLE 1> ;

READ<BUS 1> AND SAVE AS (Z);

APPLY 15V TO<BUS 2> ;

LET (D) = 3 + 2;

IF (D) = 7 THEN TERMINATE;

TURN OFF<SWITCH 1> ;

END SUBROUTINE;

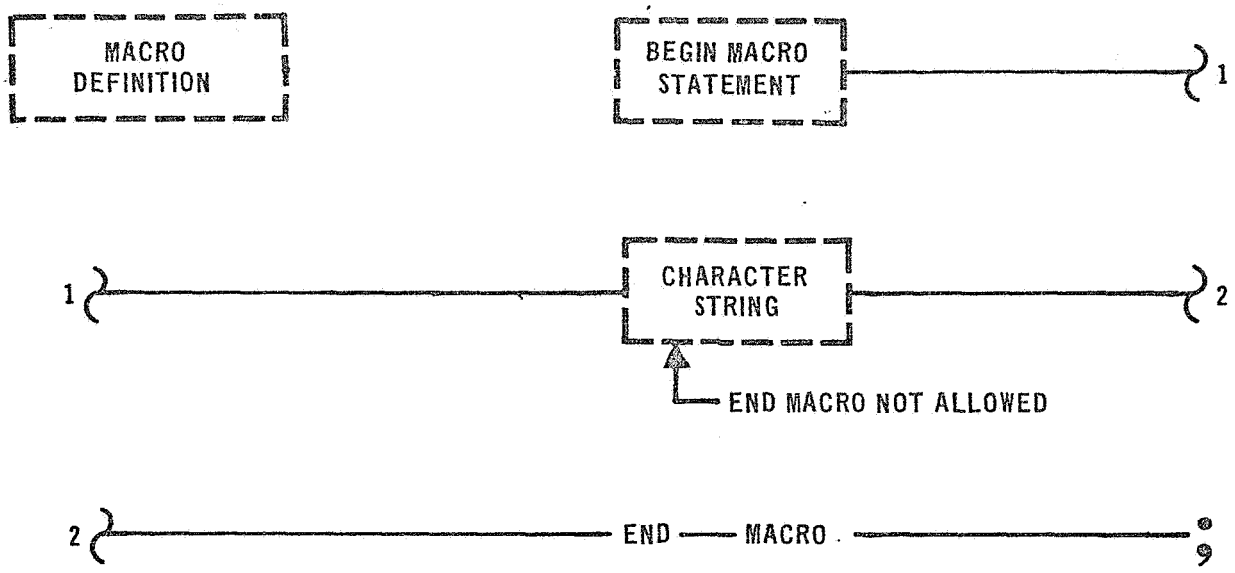
AFTER EXECUTION OF SUBROUTINE

(Z) = BUS 1 READING

(D) = 5

NOTE- (Z) DATA AVAILABLE TO PROGRAM

(D) DATA SUBROUTINE ONLY


MACRO
DEFINITION
BEGIN MACRO
STATEMENT
1
1
CHARACTER
STRING
2
END MACRO NOT ALLOWED
2
END
MACRO
;

### 5.1.4 Macro

A Macro is a named Character String. A Macro must start with a BEGIN MACRO STATEMENT and end with an END STATEMENT. The Character String, between the BEGIN MACRO STATEMENT and END STATEMENT, is defined as a Macro skeleton and is made up of any combination of GOAL characters. The only exception is the Character String "END MACRO" which is invalid.

Once the Macro is defined, each reference to the Macro Label by an EXPAND MACRO STATEMENT will cause the Macro skeleton for that Macro to be inserted into the source program. A Macro may be inserted anywhere within a Program as long as it is defined prior to its use by an EXPAND MACRO STATEMENT. By convention, Macros should be grouped together and should immediately follow the Begin Statement of the host component.

The BEGIN MACRO STATEMENT allows the test writer to assign an arbitrary Name to Parameters. The Parameters may be any legal Name and must be unique only within the Macro. In other words, a Parameter defined within a Macro may also be defined as a Name within the main program without being multiply defined. The same Parameter could also appear in several Macros and also as a Name in the main body of the Program and not be in error. A Parameter may be referenced as many times as desired within the Macro skeleton. The EXPAND MACRO STATEMENT specifies the contents, substitutable arguments, of the Parameters defined in the BEGIN MACRO STATEMENT.

The EXPAND MACRO STATEMENT allows the test writer to define the necessary Character Strings which are used to replace the corresponding substitutable arguments in a Macro skeleton. When the EXPAND MACRO STATEMENT is processed, the list of Parameters in the BEGIN MACRO

STATEMENT is linked one-to-one to the Character Strings in the EXPAND MACRO STATEMENT. The first Parameter is linked to the Character String between the first and second commas. The second Parameter is linked to the Character String between the second and third commas. This connecting process continues until all listed Parameters are linked to a specific Character String. The Language Processor then searches the Macro skeleton for substitutable arguments. When a substitutable argument is located, it is replaced by its corresponding Character String. The Language Processor continues its search of the Macro skeleton until it encounters an END MACRO STATEMENT. The Language Processor then inserts the Macro skeleton with the substituted Character String in the source Program. The printing of the Macro skeleton is controlled by the EXPAND MACRO STATEMENT. The control that the EXPAND MACRO STATEMENT has over the Macro skeleton is shown in the following examples. A Macro definition is given first and followed by an Expand option.

The Expand option prints only the Macro skeleton after the Character String has been substituted.

The Execute option is shown next. The Execute option results in the print of the Macro Label and the substituted Character Strings. The Expand and Execute option is shown last. This option results in a printing of a combination of the two other options.

5.1.4.1 Macro Example

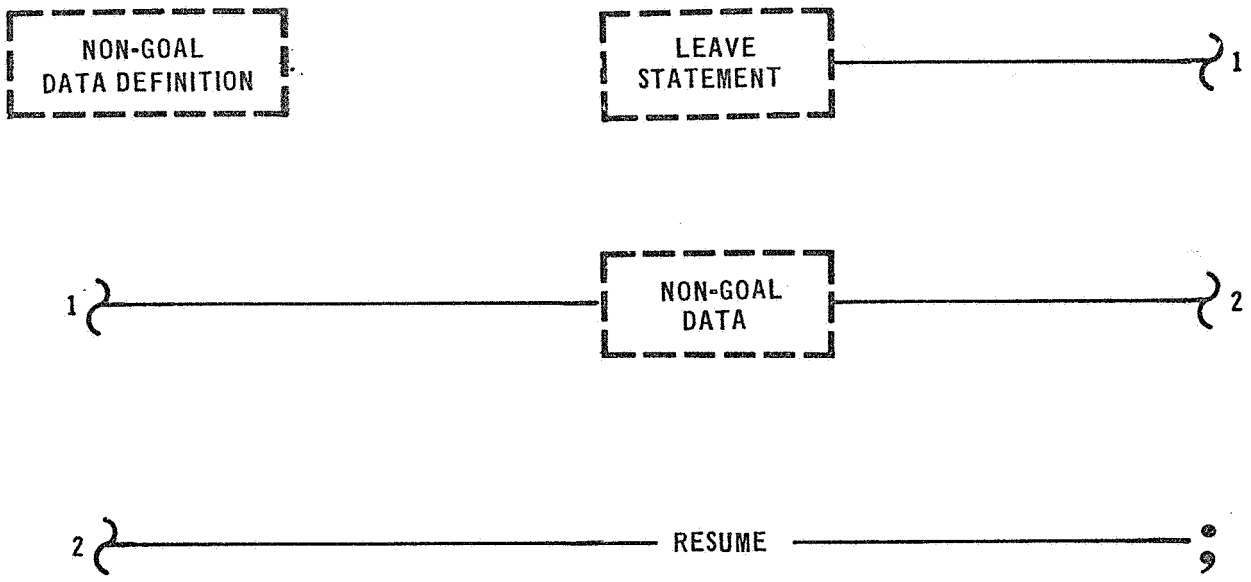
GOAL PROGRAM
(Card Listing)

```
BEGIN PROGRAM (EXAMPLE)
      REVISION 2;
BEGIN MACRO STATUS
      (A), (B);
VERIFY (A) IS OFF;
VERIFY (B) IS OFF;
END MACRO

_____

_____

EXPAND STATUS
,<PREFLT CAL ON>
,<INFLT CAL ON >,;

_____

_____

EXECUTE STATUS
,<PREFLT CAL ON>
,<INFLT CAL ON >,;

_____

_____

EXPAND AND EXECUTE
,<PREFLT CAL ON>
,<INFLT CAL ON >,;
```

GOAL PROGRAM
(Processor Listing)

```
BEGIN PROGRAM (EXAMPLE)
      REVISION 2;
BEGIN MACRO STATUS
      (A), (B);
VERIFY (A) IS OFF;
VERIFY (B) IS OFF;
END MACRO

_____

_____

VERIFY<PRELFT CAL ON> IS OFF;
VERIFY<INFLT CAL ON >IS OFF;

_____

_____

STATUS,<PREFLT CAL ON>
      ,<INFLT CAL ON >,;

_____

_____

STATUS,<PREFLT CAL ON>
      ,<INFLT CAL ON> ,;
VERIFY<PREFLT CAL ON> IS OFF
VERIFY<INFLT CAL ON >IS OFF;
```

_____ Implies one or more statements


NON-GOAL
DATA DEFINITION
LEAVE
STATEMENT
1
1
NON-GOAL
DATA
2
2
RESUME
;

### 5.1.5 Non-Goal

A NON-GOAL component consists of a package written in another language. Since it is impossible to predict all testing capabilities that are required of a test language, the GOAL test language provides for the use of other languages. The availability of other languages to be used as a NON-GOAL component is directly dependent upon the design of a total test system.

A compatible language may be used to create a NON-GOAL component by compiling the other language routine by its own compiler. The object deck of the other language is inserted in a Data Bank Subroutine. The object deck must be immediately preceded by a LEAVE STATEMENT and must be immediately followed by a RESUME STATEMENT. Such a Subroutine may also contain additional GOAL statements, if necessary.

The execution of a NON-GOAL component is controlled by a PERFORM SUB-ROUTINE STATEMENT. Once control is passed to a Subroutine any GOAL statements preceding or following the NON-GOAL component will be executed in their normal sequence.

When a LEAVE STATEMENT is encountered by the executive, control is passed to the NON-GOAL component. After execution of a NON-GOAL component, control may be returned to the Data Bank Subroutine at the RESUME STATEMENT. The remaining GOAL statements are then executed in their normal sequence. Communication between the Data Bank Subroutine and the NON-GOAL component may be achieved by passing data declared in the LEAVE STATEMENT.

## 5.2 LANGUAGE SYSTEM CONCEPTS

The LANGUAGE SYSTEM contains three major divisions. Namely, the language, the Language Processor, and the system executive. Each of these divisions overlap somewhat in function. It is sometimes very difficult to distinguish between a language function, processor function, or executive function.

### 5.2.1 Language

The language must provide statements which will contain a complete command, just as an English statement requires all of its parts to be complete. It must make sure that its statement contains information that will be recognizable by the processor, and that each statement contains enough information to enable the executive to perform the command.

The language is required to contain all the information necessary to provide a complete checkout of the system under test.

### 5.2.2 Language Processor

The Language Processor is responsible to check the language statement to verify the statement syntax. The processor must convert the program statement from its input source format into an executive-recognizable format. It must not only assure correct grammar but also that all parts of the statement that are necessary for a command to be performed are present in the statement.

The processor must link the test program to the system under test via the Data Bank. It relates statements to statements and verifies that language components are properly combined.

5.2.2.1 System Subroutines

Subroutines may be inserted between any SPECIFY STATEMENTS in the Data Bank. The Subroutines would then be available to any program at compile time.

When a system Subroutine is requested in a program by a PERFORM SUBROUTINE STATEMENT, the Language Processor will first try to find a local Subroutine and will then look for a system Subroutine. If none is found, an error condition will be flagged.

5.2.2.2 System Macros

System Macros will be available to any program via a system Macro package in the Language Processor.

When a Macro is requested in a program by an EXPAND MACRO STATEMENT, the Language Processor will first try to find a local Macro and will then look for a system Macro. If none is found, an error condition will be flagged.

5.2.2.3 Processor Options

The GOAL processor will provide several options. These include:

1. Source Listings - Listing of all source records processed.
2. Expanded Statement Listing - A listing of all statements after processing with compiler generated step numbers.
3. Statement Label Cross Reference - A list of all statement labels defined or referenced in a GOAL program.
4. Internal Name Cross Reference - A listing of all symbolic names defined or referenced in a GOAL program.
5. Function Designator Summary - A listing of all Function Designators referenced in a GOAL program.

6. Diagnostics Summary - A listing of all errors detected.

Several directives will also be provided, including:

1. Sequencing Field Size
2. Edit Only
3. List output report selection
4. Title for listing page
5. Date for listing page
6. Number of lines per page on printer listing
7. Page counter

### 5.2.3 Executive

The executive must take the output of the Language Processor and perform the necessary program functions. The executive must handle such items as scheduling events, timing for data acquisition, engineer-executive communication, etc.

Many language statements are highly implementation dependent. The result of a statement may be considerably different depending on how the executive handles the statement. It is conceivable that language statements written prior to development of an executive may have to be changed somewhat after executive development. For example, a statement is legal which says "EVERY 2 SECS CONCURRENTLY PERFORM PROGRAM (A);" but the executive may be unable to handle a 2 second cyclic rate for program execution, thus a minimum cyclic rate would have to be imposed.

## 5.3 PROGRAMMING CONCEPTS

Many concepts were legislated during the development of the GOAL

language. Five of the more special concepts are discussed in the following items: concurrent processing, language level interrupt, table techniques, program termination, and dimensions.

### 5.3.1 Concurrency

Concurrency is used within this write-up in the general context which assumes at least a second, independently compiled, test application task may be initiated prior to the completion of another task. This does not force a selection of either a multiprogramming system or a parallel processing system. The effort was made, assuming some concurrency does exist, to ascertain what information required can best be specified at test preparation time and how this information can be presented in a readable manner.

The test environment appears to dictate that a "reactive monitor" is needed. This is long term surveillance of a test condition coupled with the capability to initiate commands if the status violates limits. Conventional monitoring is embedded in this capability. The CONCURRENT STATEMENT in GOAL provides the "reactive" capability by allowing another program to be performed. The statement provides a simplified method of supplying monitor only parameters for either continuous monitoring or exception monitoring.

### 5.3.2 Language Level Interrupts

Language Level Interrupts indicate to the executive that an external event has occurred, and that this particular event requires attention. Language Level Interrupts permit key Function Designators to interrupt the execution of a test program and automatically transfer control to

a specified Interrupt response Subroutine and also to another language statement. After execution of an Interrupt response Subroutine, control is returned to the calling program. Language Level Interrupts are not to be confused with System Interrupts, that control the execution of the executive during the handling of such items as timing or communication. The term Interrupt in this document refers to a Language Level Interrupt if not otherwise stated.

A Function Designator can interrupt a test program only if the following three (3) steps are performed.

(1) The Function Designator must be declared as an Interrupt within the Data Bank. The necessary conditions for the interruption to occur must also be defined in the SPECIFY STATEMENT.

(2) A WHEN INTERRUPT STATEMENT that references the Interrupt Function Designator must be previously executed. The WHEN INTERRUPT STATEMENT enables the Interrupt. It also defines where control is to be passed at the occurrence of the Interrupt.

(3) Finally, the conditions specified for the Interrupt must be met and the Interrupt Function Designator must still be enabled and not suspended. The execution of the test program will then be interrupted for servicing of the Interrupt.

An Interrupt enabled for a particular program is suspended during execution of a Subroutine. That is, if an Interrupt enabled in a

program occurs during the execution of a Subroutine, the Interrupt will not be serviced until control is passed to the calling program.

An Interrupt that occurs during the execution of a Subroutine will be serviced only if the Interrupt was enabled by the Subroutine. An Interrupt enabled by both the program and Subroutine that occurs during the execution of the Subroutine will first be serviced during execution of the Subroutine. If the Interrupt is still active when control is returned to the calling program, it will be serviced for the second time during the execution of the test program.

The WHEN INTERRUPT STATEMENT enables an Interrupt and the DISABLE INTERRUPT STATEMENT disables the Interrupt. If an Interrupt occurs after it has been disabled, it will not be serviced, the execution of the test program will not be interrupted by the occurrence of an Interrupt that is disabled.

The WHEN INTERRUPT STATEMENT may also specify a Statement Number instead of a Subroutine. In this case, control will be passed to the indicated statement at the time of an Interrupt occurrence.

The following Examples show various types of Interrupt conditions.

5.3.2.1 Language Level Interrupt Examples

| PROGRAM A | SUBROUTINE SA |
|---|---|
| S10 WHEN INTERRUPT <X> OCCURS | S5 _____ |
| GO TO STEP 500; | _____ |
| S40 | |
| S70 WHEN INTERRUPT <X> OCCURS | S10 WHEN INTERRUPT <X> OCCURS |
| GO TO STEP 700; | GO TO STEP 20; |
| S90 | _____ |
| S105 PERFORM SUBROUTINE (SA); | _____ |
| S200 _____ | S15 LET (A) = (B); |
| _____ | _____ |
| _____ | S20 SET <SYSTEM FLAG> TO ON; |
| S500 OPEN <VENT VALVES>; | S22 DISABLE STEP 10; |
| TERMINATE; | _____ |
| S700 TURN OFF <SYSTEM POWER>; | _____ |
| TERMINATE; | S30 ASSIGN (K) = OFF; |
| _____ | S36 _____ |
| _____ | _____ |
| | TERMINATE; |
| | _____ |

_____ IMPLIES ONE OR MORE STATEMENTS

INTERRUPT ASSUMPTIONS

Example 1.

CONDITION:

EXECUTING IN "A"

- STEP 10 PREVIOUSLY EXECUTED AND STILL ENABLED.
- STEP 40 IS BEING EXECUTED WHEN INTERRUPT <X> OCCURS.

RESPONSE:

- STEP 40 IS COMPLETED
- EXECUTION IS RESUMED AT STEP 500.

Example 2.

CONDITION:

"A" IS EXECUTING

- STEP 10 IS EXECUTED
- STEP 70 IS EXECUTED
- STEP 90 IS BEING PROCESSED WHEN INTERRUPT <X> OCCURS.

RESPONSE:

- STEP 90 IS COMPLETED
- EXECUTION IS RESUMED AT STEP 700;

COMMENT:

THE EXECUTION OF STEP 70 HAD THE AFFECT OF DISABLING STEP 10 BECAUSE THEY REFERENCED THE SAME INTERRUPT.

Example 3.

CONDITION:

- STEP 70 IN "A" PREVIOUSLY EXECUTED.
- "SA" IS EXECUTING AS A RESULT OF STEP 105 IN PROGRAM A.

Example 3 (CTD):

- STEP 5 IS BEING EXECUTED WHEN INTERRUPT <X> OCCURS.

RESPONSE:

- "SA" CONTINUES NORMAL PROCESSING

COMMENT:

WHEN CONTROL IS RETURNED TO PROGRAM A AT THE STEP IMMEDIATELY FOLLOWING STEP 105, THEN THE INTERRUPT ACTION IS NOTED AND CONTROL DIVERTED TO 700.

Example 4.

CONDITION:

- STEP 70 IN "A" PREVIOUSLY EXECUTED.
- "SA" IS EXECUTING AS A RESULT OF STEP 105 IN PROGRAM A.
- STEP 10 IN "SA" PREVIOUSLY EXECUTED.
- STEP 15 IS EXECUTING WHEN INTERRUPT <X> OCCURS.

RESPONSE:

- STEP 15 COMPLETES PROCESSING.
- STEP 20 IN "SA" IS THE NEXT STEP PROCESSED.

COMMENT:

WHEN CONTROL IS RETURNED TO PROGRAM A AT THE STEP IMMEDIATELY FOLLOWING STEP 105, THEN THE INTERRUPT STATUS OF <X> IS TESTED AND IF STILL ACTIVE CONTROL IS RESUMED AT STEP 700.

Example 5.

CONDITION:

- STEP 70 IN "A" PREVIOUSLY EXECUTED.
- "SA" IS EXECUTING AS A RESULT OF STEP 105 IN PROGRAM A.

Example 5 (CTD):

- STEP 22 IN "SA" PREVIOUSLY EXECUTED.
- STEP 36 IN "SA" *IS BEING EXECUTED WHEN INTERRUPT* <X> *OCCURS.*

RESPONSE:

- "SA" CONTINUES NORMAL PROCESSING.

COMMENT:

THE INTERRUPT HAD NO AFFECT ON "SA" AS STEP 22 HAD CANCELLED THE REQUEST FOR INTERRUPT ACTION. EVEN THOUGH THE STEP 22 DISABLES STEP 10, THIS IS STEP 10 WITHIN "SA" ONLY.

### 5.3.3 Table Techniques

Tables are especially suited to test applications involving test patterns such as those required in the testing of voting logic and interface connections.

The Declaration of tables is discussed in Section II.

The general format of a table is depicted below.

| | TABLE NAME | | | | |
|---|---|---|---|---|---|
| | | COLUMN NAME | COLUMN NAME | COLUMN NAME | COLUMN NAME |
| Row 1 | FUNCTION DESIGNATOR | data | data | data | data |
| Row 2 | FUNCTION DESIGNATOR | data | data | data | data |
| Row 3 | FUNCTION DESIGNATOR | data | data | data | data |
| Row Rn | FUNCTION DESIGNATOR | data | data | data | data |
| | | 1 | 2 | 3 | Cn |

Column Numbers

Note that row numbers and columns are not part of the table itself but may be used by statements referencing the table to locate data or data storage. Any reference to non-existent rows by using the Row Number feature will be flagged as an error at processing time.

If by an erroneous calculation an Index Name contains a reference to non-existent data location, then the executive system should indicate the error. This applies for columns also.

If the same Function Designator appears more than once in the table outline, then the entries are flagged as possible errors. If an ACTIVATE TABLE STATEMENT or INHIBIT TABLE STATEMENT later uses this multiple entered Function Designator as a Row Designator, then the processor will direct the action to be taken on all rows with that Function Designator.

```
DECLARE STATE TABLE (ENGINE LOGIC) WITH 5 ROWS AND 3 COLUMNS
        TITLED
                                (PASS 1), (PASS 2), (PASS 3) WITH ENTRIES
<ENG 1 IGN PHASE
SOLENOID> ;                      ON,      ON,      ,
<ENGINE NO 1
CUTOFF> ,                        ON,      OFF,     ,
<START TANK ALL
VENT> ,                          OFF,     OFF,     ,
<ENG 1 ST TANK
DISCHARGE CONTROL> ,             ON,      ON,      ,
<MAIN STAGE CONTROL
SOLENOID> ,                      ON,      OFF,     ;
```

Example 1: TURN ON (ENGINE LOGIC) FUNCTIONS;

Example 2: SET (ENGINE LOGIC) FUNCTIONS TO (PASS 1);

Example 3: READ (ENGINE LOGIC) FUNCTIONS AND SAVE AS (PASS 3);

Example 4: VERIFY (ENGINE LOGIC) FUNCTIONS ARE (PASS 2);

DECLARE QUANTITY TABLE (BUS READINGS) WITH 4 ROWS AND 4 COLUMNS WITH ENTRIES

<BUS 10>, 10V, 15V, , 14V,
<BUS 20>, 15V, 20V, , 12V,
<BUS 30>, 12V, 18V, , 25V,
<BUS 40>, 14V, 20V, , 16V;

Example 1: VERIFY (BUS READINGS) FUNCTIONS ARE COLUMN 2;

Example 2: VERIFY (BUS READINGS) FUNCTIONS ARE BETWEEN COLUMN 1 AND COLUMN 2;

Example 3: INHIBIT (BUS READINGS) ROW 3;
VERIFY (BUS READINGS) FUNCTIONS ARE LESS THAN COLUMN 3;

Example 4: READ <BUS 20> AND SAVE AS (BUS READINGS) ROW 3 COLUMN 3;

If the target hardware system provides special facility to group commands, then it is assumed the processor will package the commands together providing all the functions for the table belong to the same group. This is an implementation option. The functions will otherwise be issued in a serial manner starting with Row 1 and proceeding to Row N. Rows that are inhibited will be ignored.

```
DECLARE STATE TABLE (POWER) WITH 4 ROWS AND 3 COLUMNS TITLED
                              (MAIN), (BACK UP), (INITIAL STATE)
        WITH ENTRIES
$ ROW 1;<MAIN POWER 1>     , ON,      OFF,          ,
$ ROW 2;<MAIN POWER 2>     ,  ON,     OFF,          ,
$ ROW 3;<BACK UP POWER 1>  , OFF,      ON,          ,
$ ROW 4;<BACK UP POWER 2>  ,  OFF,     ON,          ;
DECLARE TEXT LIST (POWER MESSAGE LIST) WITH 4 ENTRIES
$ MESSAGE NO 1; (MAIN POWER SUPPLY NO 1 IS NOT ON),
$ MESSAGE NO 2; (MAIN POWER SUPPLY NO 2 IS NOT ON),
$ MESSAGE NO 3; (BACK UP POWER SUPPLY NO 1 IS ON),
$ MESSAGE NO 4; (BACK UP POWER SUPPLY NO 2 IS ON);
```

Example: VERIFY (POWER) FUNCTIONS ARE EQUAL TO (MAIN) ELSE DISPLAY EXCEPTIONS USING MESSAGES FROM (POWER MESSAGE LIST) AND STOP;

The message list must be as long as, or longer than, the power table. The exception is that one message may be used for many exceptions by explicitly denoting or referencing one message.

When the Table Name is used without row/column modification, then the action is taken for all active functions in the table. Also note that (INITIAL STATE) may be a column name in the (POWER) table in which case the rewriting of the Table Name is optional. When the example is executed, a reading is performed to obtain the latest available data and then the comparison is made. The order starts with Row 1. If an item fails to pass the Comparison Test, then the corresponding message list entry is displayed; e.g., if entry 4 in the (POWER) table failed, then message 4 is displayed. Message correlation is done regardless if any of the preceding rows were inhibited or not.

### 5.3.4 Program Termination

System assumptions were made for program termination capability for serial and concurrent processing. Two termination options are provided: "TERMINATE" and "TERMINATE SYSTEM."

Serial Processing:

Consider PROGRAM A performs PROGRAM B. If a TERMINATE is executed in PROGRAM B, then processing of PROGRAM B ceases and control is returned to PROGRAM A and PROGRAM A continues. However, if in the above example, a TERMINATE SYSTEM had been encountered in PROGRAM B instead of a normal TERMINATE, then not only would PROGRAM B be stopped but also the chain of execution through which execution was initiated, in this case, PROGRAM A. The effect would be the same if instead of PROGRAM B, it was a Subroutine that PROGRAM A had executed.

Concurrent Processing:

Now assume PROGRAM A performs PROGRAM B concurrently with a five minute cycle rate. If a TERMINATE is executed in PROGRAM B, then PROGRAM B stops execution and remains stopped until the concurrency rate requirements are met and then program B is executed again.

Using the same example, assume a TERMINATE SYSTEM was encountered. The program stops execution and also the request for concurrent execution is cancelled. PROGRAM B would not be automatically recalled by the system. PROGRAM A would not be terminated. If PROGRAM A was dependent on PROGRAM B's continued execution, then care should have been taken for PROGRAM B to signal PROGRAM A, probably by a system indicator, before executing the TERMINATE SYSTEM.

5.3.5 Dimensions

Dimensions are allowed to promote the readability of the statements using or referencing the data associated with the given Dimension. Standard abbreviations are allowed and are validated by the Language Processor. The test writer must take the necessary precautions to insure the consistent use of compatible Dimensions by procedural statements. The processor/executive is not expected to perform compatibility checks. A given system may elect to implement a processor that recognizes certain engineering units as scaling directives to the processor but to proceed much further appears unlikely at this time.

If the full name of the basic unit Dimension is used, then either plural or singular will be allowed.

DIMENSION TABLE Engineering units available for use in GOAL are listed in the following matrix.

| FUNCTION TYPE | BASIC UNIT | $X10^0$ | $X10^3$ | $X10^6$ | $X10^{-3}$ | $X10^{-6}$ |
|---|---|---|---|---|---|---|
| volts ac/dc | volt | V | | | MV | UV |
| current ac/dc | ampere | A | | | MA | UA |
| frequency | hertz | HZ | KHZ | MHZ | | |
| | pulses per second | PPS | KPPS | | | |
| time | day | DAY | | | | |
| | hour | HR | | | | |
| | minute | MIN | | | | |
| | second | SEC | | | MSEC | USEC |
| resistance | ohm | OHM | KOHM | MOHM | | |
| inductance | henry | H | | | MH | UH |
| capacitance | farad | FD | | | | UFD |
| power | watt | W | KW | | MW | UW |
| | voltage, current or power | DB | | | | |
| ratio | percent | PCT | | | | |
| pressure | pounds per square inch | PSIG<br>PSIA<br>PSI | | | | |
| | millimeters of mercury | MMHG | | | | |
| | inches of mercury | INHG | | | | |
| | millibars | MB | | | | |
| distance | inch | IN | | | | |
| | foot | FT | | | | |
| | meter | M | KM | | MM | |
| | nautical mile | NM | | | | |
| velocity | feet per second | FT/SEC | | | | |
| | meters per second | M/SEC | | | | |
| | knot | KT | | | | |
| | mach no. | MACH | | | | |
| angle | degree | DEG | | | | |
| | arcmin | ARCMIN | | | | |
| | arcsec | ARCSEC | | | | |
| | radian | RAD | | | MRAD | |
| | revolution | REV | | | | |
| temperature | degrees centigrade | DEGC | | | | |
| | degrees fahrenheit | DEGF | | | | |

CAUTION - The writer must take necessary precautions to insure the consistent use of compatible dimensions.

Other allowable dimensions are:

| | |
|---|---|
| KILOVOLTS (AC or DC) | KV |
| DECIBELS above one milliwatt | DBM |
| DECIBELS above one watt | DBW |
| KILOVOLT AMPERES | KVA |
| VOLT AMPERES REACTIVE | VAR |
| KILOVOLT AMPERES REACTIVE | KVAR |
| PICOFARADS | PFD |
| MASS (grams) | G |
| ACCELERATION | M/SEC/SEC<br>FT/SEC/SEC |

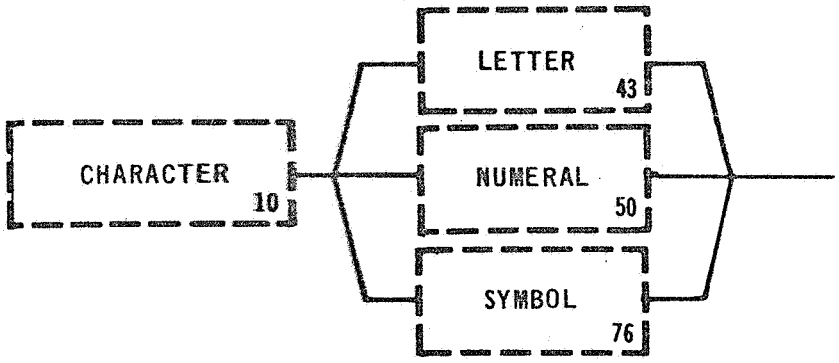
# SECTION VI, GOAL ELEMENTS

## 6.0 GENERAL

This section discusses the elements which are the most basic syntax groupings of GOAL.

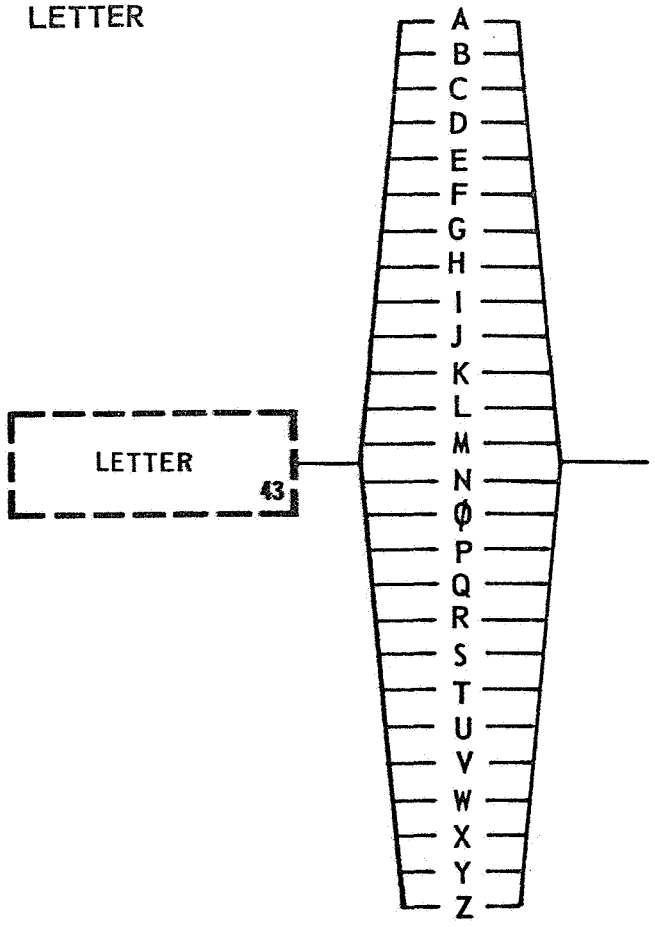
10
REV 0

CHARACTER




43
REV 0

LETTER



## 6.1 CHARACTER SET

These elements define the basic language Character Set. All language statements or elements can be traced to a basic makeup consisting of these elements.

### 6.1.1 Character

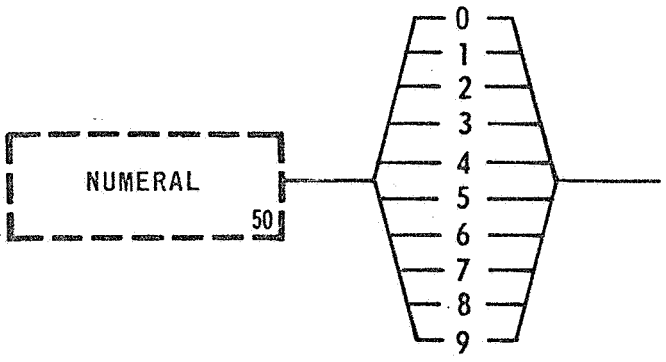
The Character element diagram is a composite showing that the full GOAL Character Set is made up of Letters, Numerals, and Symbols. It provides a cross-reference to the basic language characters.

### 6.1.2 Letter

The Letter element defines the letters available in the GOAL Character Set. These are all letters A-Z. They must always be written in the Upper Case. By convention the letter "O" should be slashed "Ø" and the letter "I" should be written as "I", and not as "l".

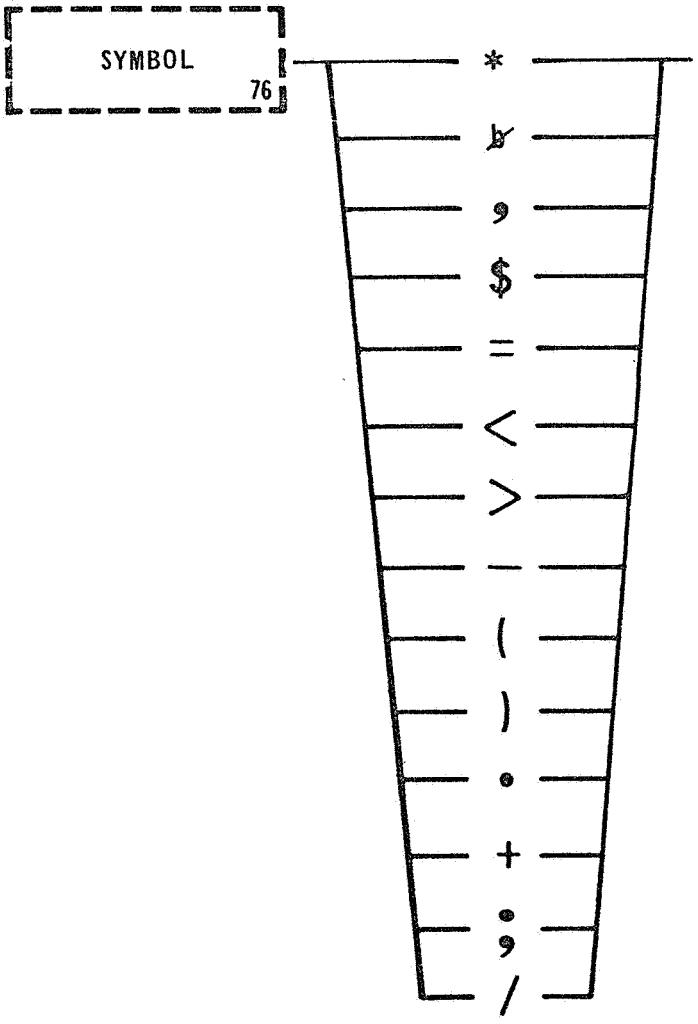
50
REV 0

## NUMERAL




76
REV 0

## SYMBOL




ASTERISK

BLANK

COMMA

CURRENCY

EQUALS

LEFT ANGLE BRACKET

RIGHT ANGLE BRACKET

MINUS

LEFT PARENTHESIS

RIGHT PARENTHESIS

PERIOD

PLUS

SEMICOLON

SLASH

### 6.1.3 Numeral

The Numeral element defines the Numerals available in the GOAL Character Set. The allowable Numerals are 0-9.

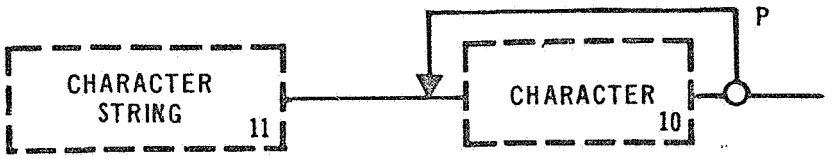
The Numeral, zero, "0" should not be slashed and the Numeral one, "1", should be written as "1" and not as "I".

### 6.1.4 Symbol

The Symbol element defines the special Symbols available in the GOAL Character Set. All special Symbols used in the GOAL Character Set are standard to the (ASCII) USA Standard Code for Information Interchange Code and the (EBCDIC) Extended Binary Code Decimal Interchange Code.

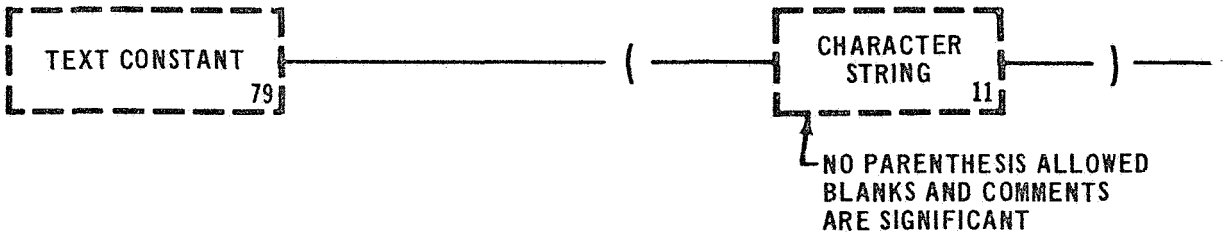
11
REV 0

## CHARACTER STRING




79
REV 0

## TEXT CONSTANT



## 6.2 CHARACTER GROUPS

These elements show how the basic characters may be linked together. These text strings are of two types: (1) an undelimited Character String, and (2) a delimited Character String (Text Constant) which is given definite syntax attributes.

### 6.2.1 Character String

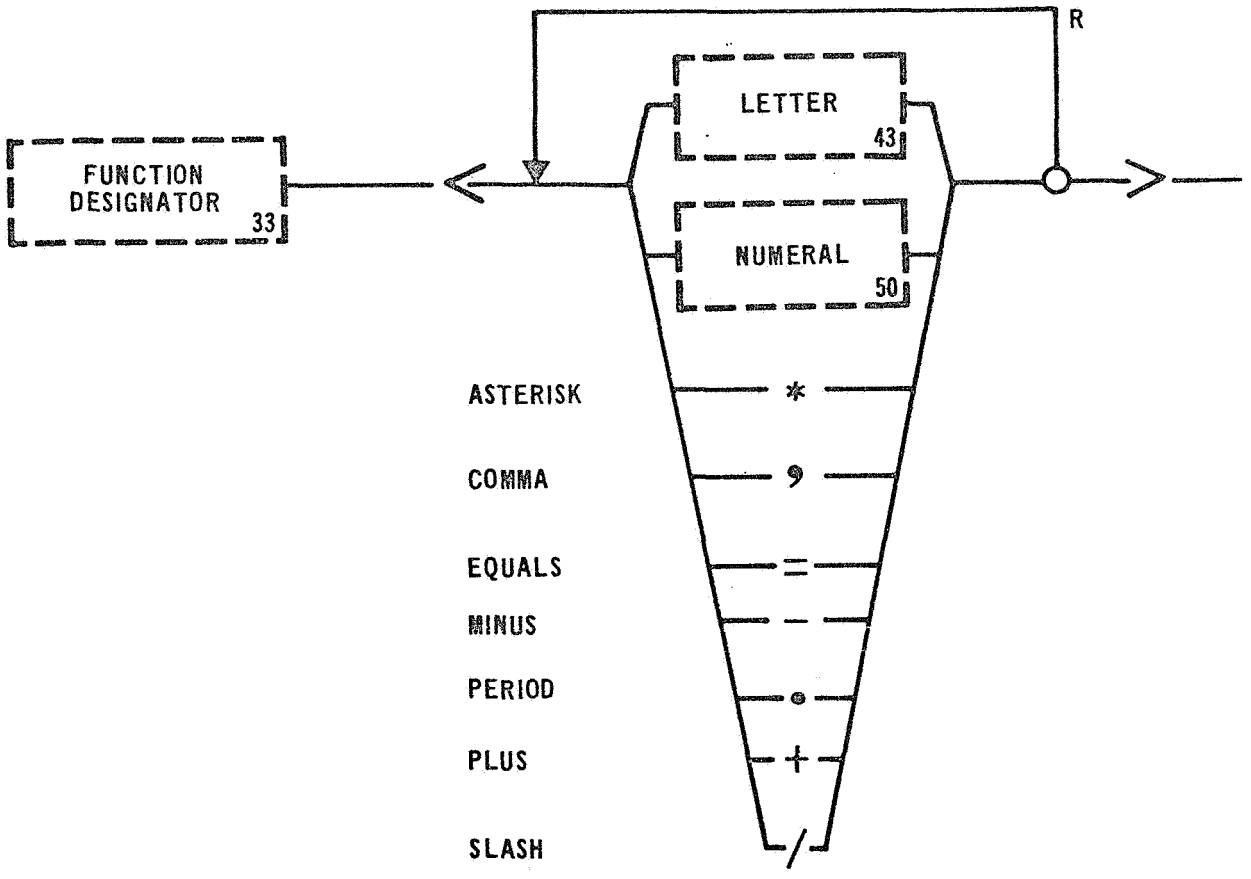
The Character String element allows Letters, Numerals, and Symbols to be placed side by side forming a string of characters.

### 6.2.2 Text Constant

The Text Constant element defines a block of Text data. It is delimited by parenthesis. Blanks are significant and could be used for formatting the Text Constant. Comments, per se, are significant in the Text Constant and if used will become a part of the Text Constant.

33
REV 0

# FUNCTION DESIGNATOR



## 6.3 EXTERNAL REFERENCE

These elements include syntactical units defining Data Bank Function Designators, and provide elements to use in referencing these Function Designators.

### 6.3.1 Function Designator

Function Designators are items which interface, via the Data Bank, with the System Under Test. All hardware linkages must be supplied by the Data Bank at compile time. Function Designators are delimited by angle brackets or "greater than" and "less than" Symbols. They are composed of either Letters, Numerals, or a Symbol subset.

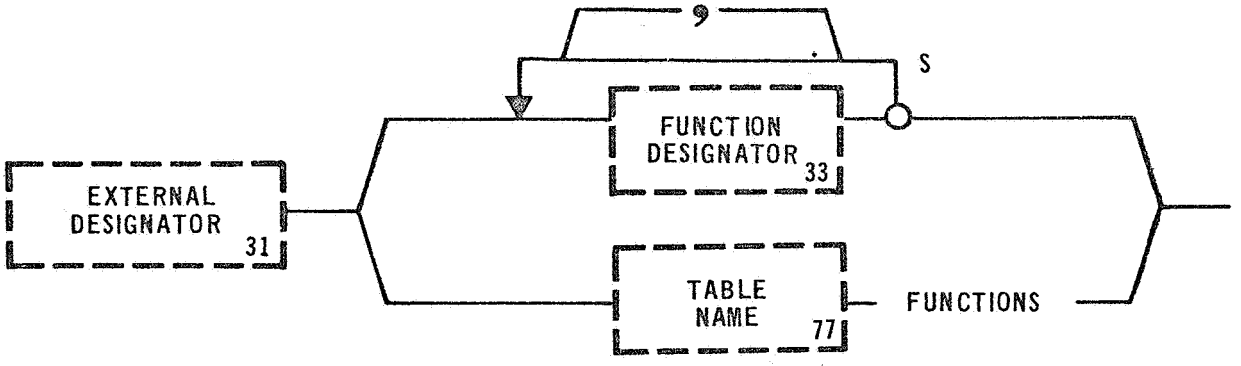
Function Designators are uniquely defined in a Data Bank via a SPECIFY STATEMENT.

Blanks and Comments are not significant within the angle brackets, therefore, care should be taken to ensure that the non blank characters form a unique string with respect to other Function Designators.

Example: <LOX VENT VALVE> <POWER TRANSFER COMPLETE IND.>
<III COOLANT TEMPERATURE> <LH2 VALVE 3 (H/L)>

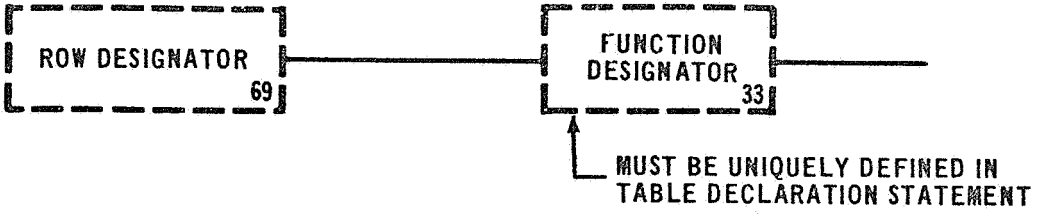
31
REV 1

## EXTERNAL DESIGNATOR




69
REV 0

## ROW DESIGNATOR



### 6.3.2 External Designator

When a statement references the External Designator syntax element the procedure writer has an option of referencing either a single Function Designator or a group of Function Designators.

When used as a group these Function Designators, which are linkages to test equipment external to the computer, must have been placed in a table. Therefore to reference a group of Function Designators it is necessary to give the table name and the word Functions. The word Functions serves as an indicator to the Language Processor that the Function Designators listed in the table will be the recipient of the stated action.

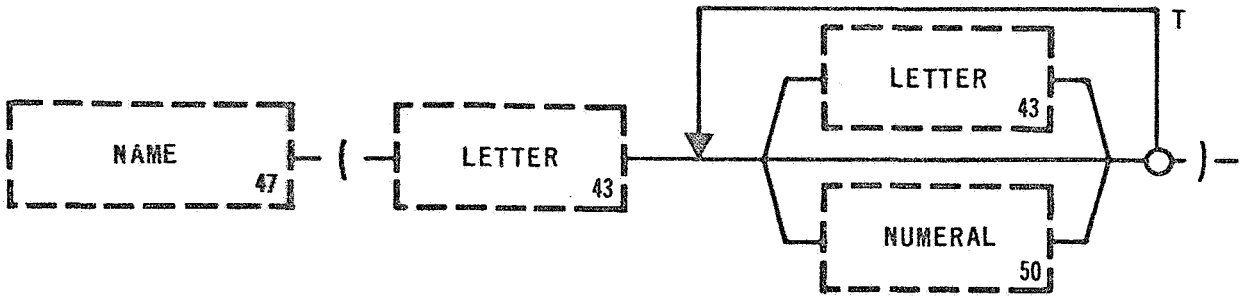
Example: <LOX VENT VALVE>
(SWITCH POSITION) FUNCTIONS

### 6.3.3 Row Designator

A Row Designator provides a means of referencing a row in a table by using the Function Designator in the Function column of that row. The Row Designator is defined in the Table Declaration Statement.

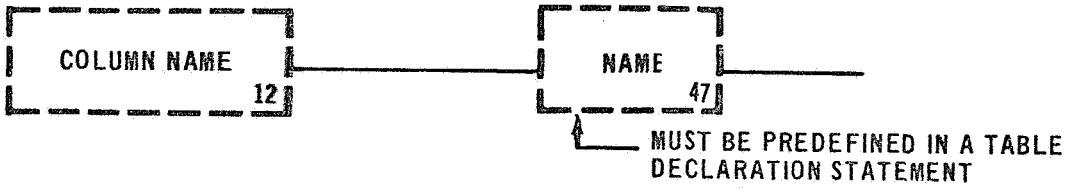
47
REV 0

NAME




12
REV 0

COLUMN NAME



## 6.4 INTERNAL REFERENCE

These elements include syntactical units defining variables that are internal to the test program and not directly related to the system under test.

### 6.4.1 Name

A Name is an internal variable defined at coding time. A Name may be used as a flag or as a storage area. It is not accessible by any other program or subroutine and is, therefore, local to the host component. All Names must be unique in any one GOAL component. A Name must begin with a Letter and may consist of a combination of Letters and Numerals only. Names are delimited by parenthesis. Neither blanks nor Comments are significant and may be used freely within the Name.

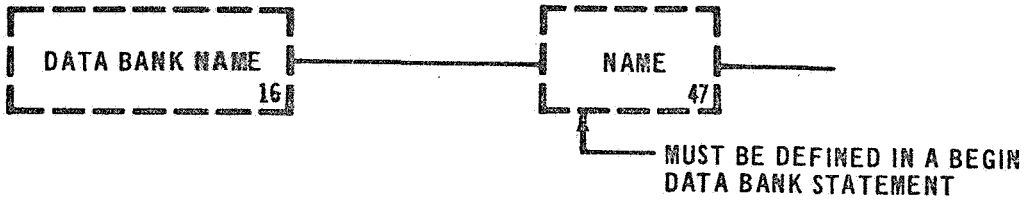
Examples: (DEGREES OF PITCH)
(TM CAL MODE RESULTS)

### 6.4.2 Column Name

A syntactic unit defined in a Table Declaration Statement and used to name a column that is part of the table being defined.

16
REV 0

## DATA BANK NAME

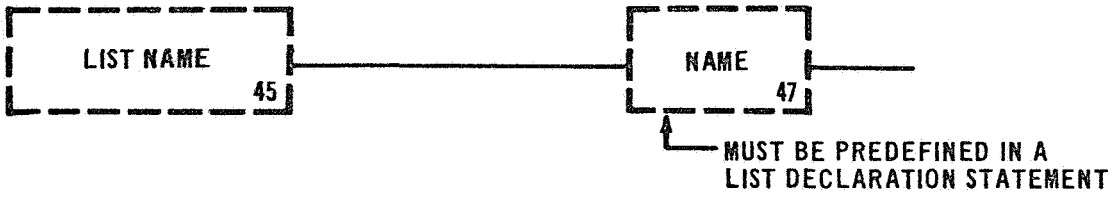



36
REV 0

## INDEX NAME

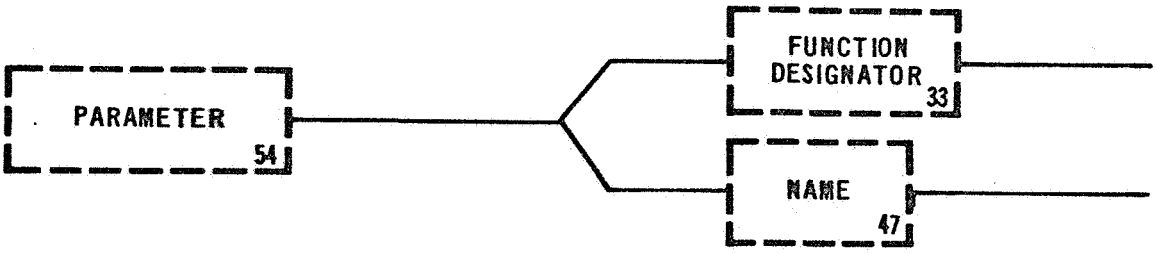

45
REV 0

## LIST NAME




54
REV 1

## PARAMETER



### 6.4.3 Data Bank Name

A syntactic unit defined in a BEGIN DATA BANK STATEMENT used to name the Data Bank component being generated.

### 6.4.4 Index Name

An Index Name must be declared as numeric in a simple Data Declaration Statement. The Index Name may be incremented or decremented by the test writer using the LET EQUAL STATEMENT.

### 6.4.5 List Name

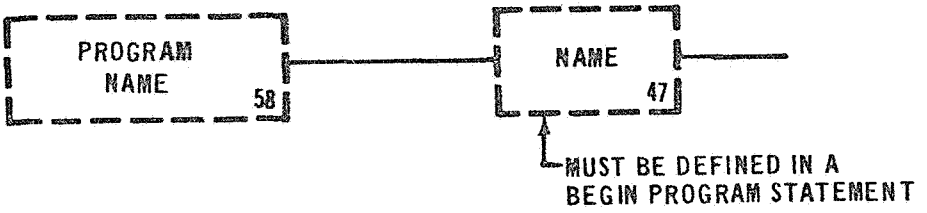
A syntactic unit indicating the name of a List and defined in a List Declaration Statement. If a List Name is used without an Integer Number or an Index Name it refers to all variables in the List.

### 6.4.6 Parameter

A Parameter represents arguments to be reserved as pseudo entries in a Macro or Subroutine, or to signify the passing of data to the Macro or Subroutine using the EXPAND MACRO STATEMENT and PERFORM SUBROUTINE STATEMENT. Parameters in the Macro are effectively text substitutions only, but Parameters in the Subroutine represent either an Internal Name or a Function Designator. Subroutine Parameters may be delimited by parenthesis or angle brackets, as appropriate. A Macro Parameter can only be delimited by parenthesis.

58
REV 0

## PROGRAM NAME

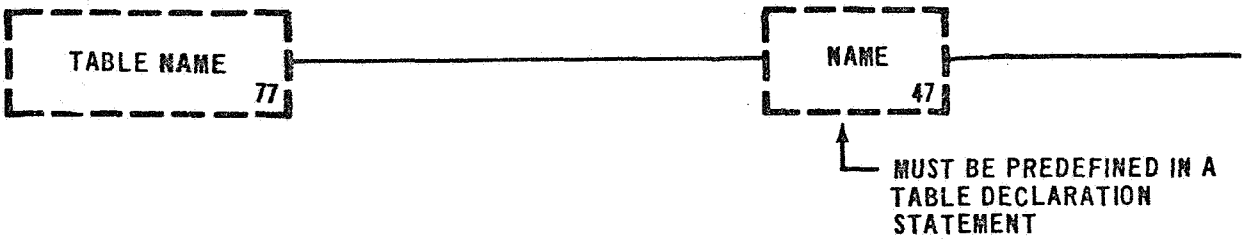



75
REV 0

## SUBROUTINE NAME


77
REV 0

## TABLE NAME



6.4.7 Program Name

A syntactic unit defined in a BEGIN PROGRAM STATEMENT used to name the Program.

6.4.8 Subroutine Name

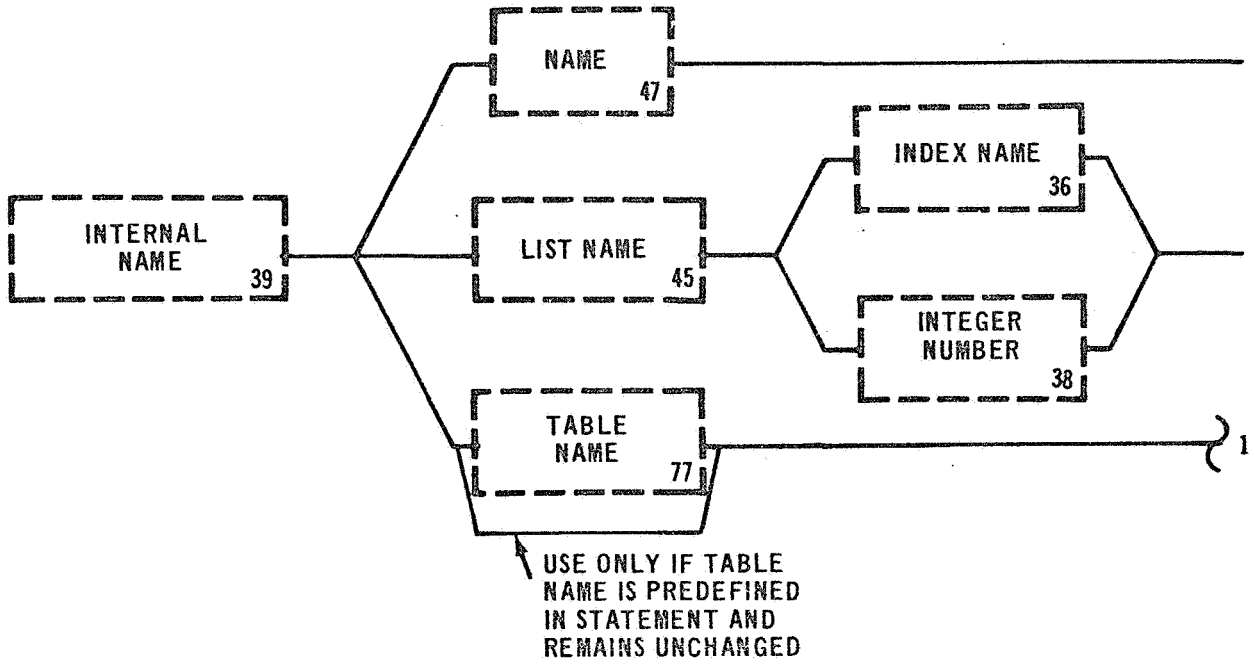
A syntactic unit defined in a BEGIN SUBROUTINE STATEMENT used to name a Subroutine.

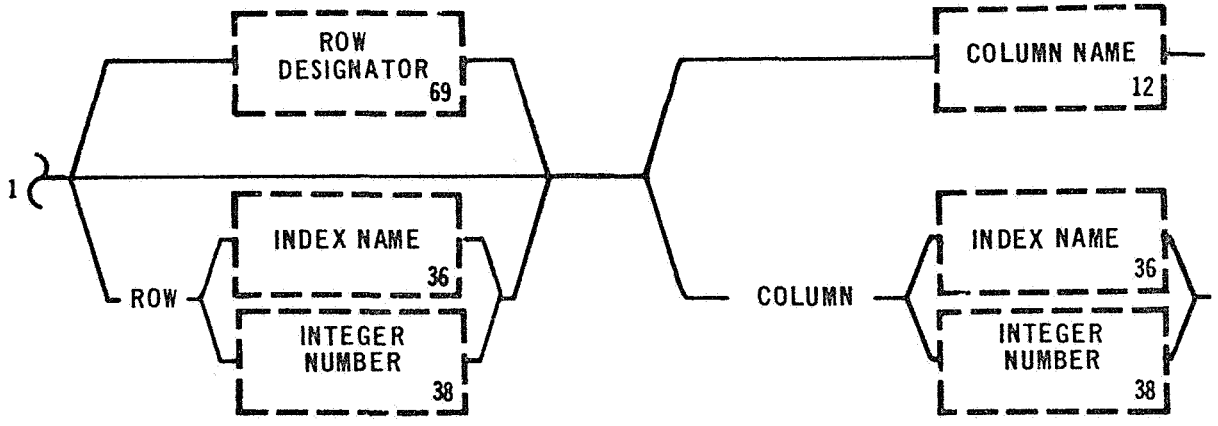
6.4.9 Table Name

A syntactic unit defined in a Table Declaration Statement used to name the Table being declared.

39
REV 1

## INTERNAL NAME



6.4.10 Internal Name

Examples: (PRESSURE READING)

(MESSAGE LIST)

(SWITCH TABLE) ROW 3 COLUMN 4

(POWER) < LAMP 1 > (RESET)

An Internal Name may be one of three types -

A. NAME - an internal variable arbitrarily chosen by the test writer.

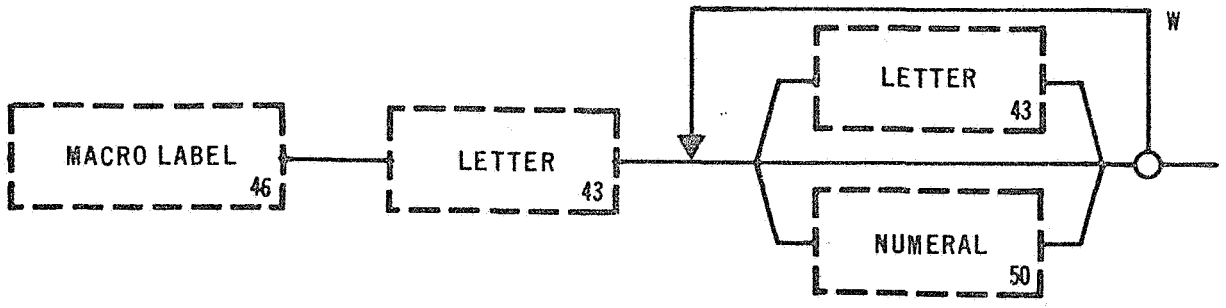
B. LIST NAME - defined in a LIST DECLARATION STATEMENT and may include indexing.

C. TABLE NAME - defined in a TABLE DECLARATION STATEMENT and must contain either an indicator for ROW and COLUMN or at least for a COLUMN. Indexing on ROWS and COLUMNS is also available.

An Internal Name is made up of Letters and Numerals only. The first character must be a Letter. Blanks and Comments are insignificant. An Internal Name is delimited by parenthesis.

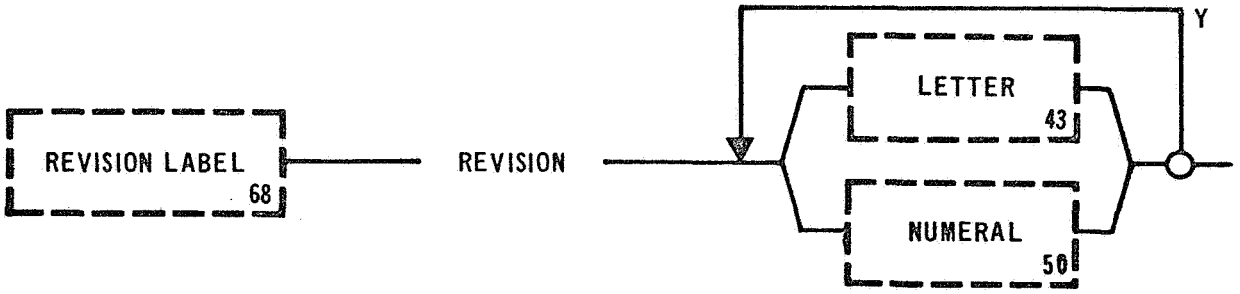
46
REV 0

## MACRO LABEL




68
REV 0

## REVISION LABEL



## 6.5 LABELS

These elements include the syntactical units that are used for processor and executive identification of a Macro, Program, and/or Data Bank.

### 6.5.1 Macro Label

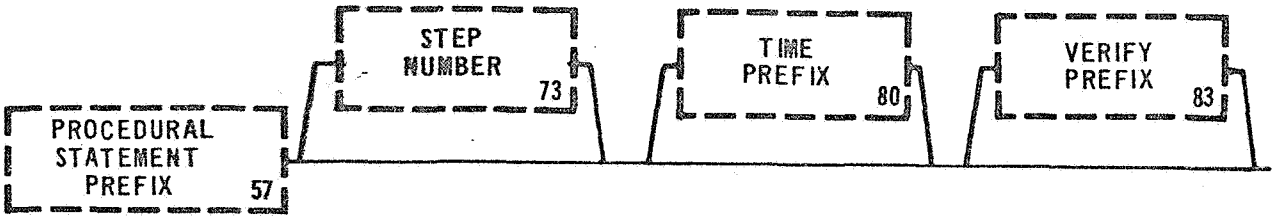
A Macro Label is defined in a BEGIN MACRO STATEMENT. A Macro Label names a combination of language statements that are created as extensions to the language and are referenced in a procedure by the name indicated by Macro Label. Parenthesis are not required as delimiters. The first character must be a Letter. The Macro Label cannot contain any symbols. The Macro Label allows free use of blanks.

### 6.5.2 Revision Label

The Revision Label indicates the revision of the GOAL component defining it in that component's Begin Statement. It may be any combination of Letters and Numerals.

57
REV 1

## PROCEDURAL STATEMENT PREFIX



## 6.6 OPTIONAL PREFIXES

These elements define the syntactical units that may be combined with any Procedural Statement. It also shows how the prefix elements may be linked together.

### 6.6.1 Procedural Statement Prefix

Examples: STEP 65 WHEN<CLOCK> IS -5 MINS,

STEP 43 AFTER<CLOCK> IS -5 MINS THEN

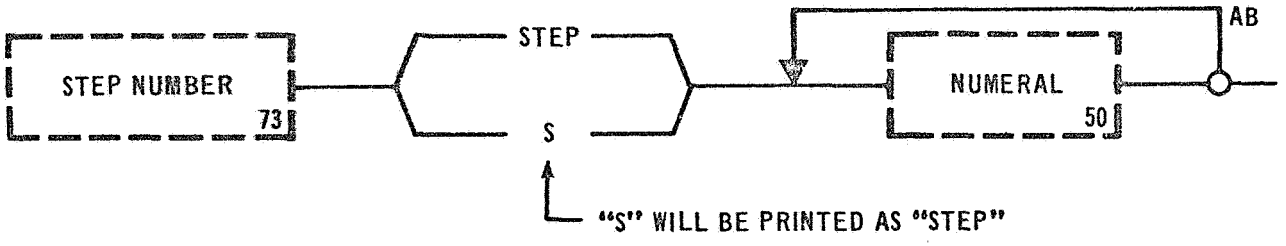
VERIFY<LOX VENT VALVE> IS OPEN,

IF (PRESS A) IS GREATER THAN 1500 PSIA THEN

The PROCEDURAL STATEMENT PREFIX allows Procedural Statements to be preceded by a combination of either Step Number and/or Time Prefix and/or Verify Prefix. This allows a Procedural Statement to be referenced by another Procedural Statement, to be executed dependent upon a time condition; and/or to be executed dependent upon an event occurrence.

73
REV 1

## STEP NUMBER



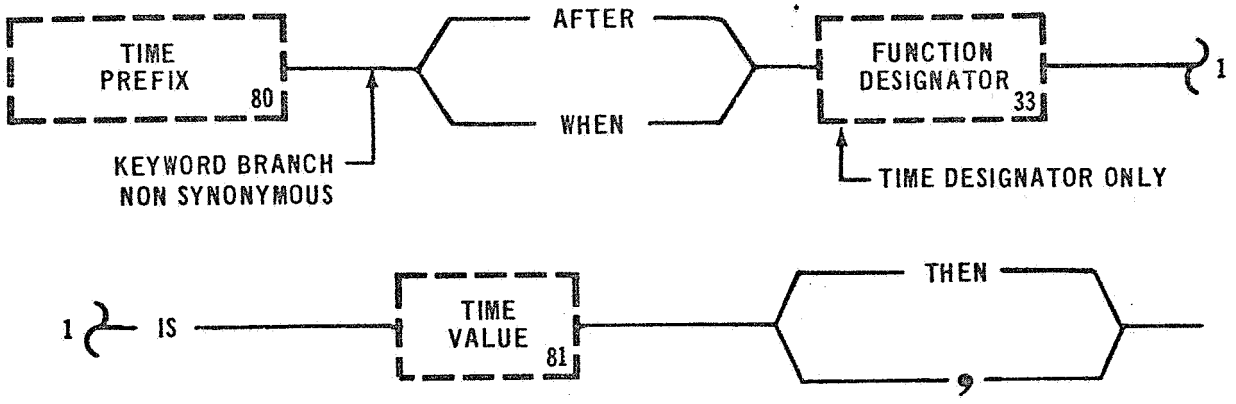
### 6.6.2 Step Number

A Step Number may be used with any Procedural Statement. It provides a means of labeling or referencing a statement. The Step Number is comprised of Numerals only and is preceded by either an "S", or "STEP". If an "S" is used at coding time, then the compiler will replace that with "STEP" at compile time. Only statements which are referenced by other statements need have a Step Number. Step Numbers are not required to be in any particular order. However, an ascending sequence convention is recommended.

Step Numbers, when used to reference a specific Procedural Statement, must reference a statement within the boundaries of the same GOAL component. For example, a Step Number in a program cannot reference a statement in a Subroutine of another Program. Step Numbers are therefore local to a component.

80
REV 2

## TIME PREFIX



### 6.6.3 Time Prefix

Examples: AFTER <CLOCK> IS -14 HRS 30 MINS,

WHEN <GMT> IS 1130 HRS THEN

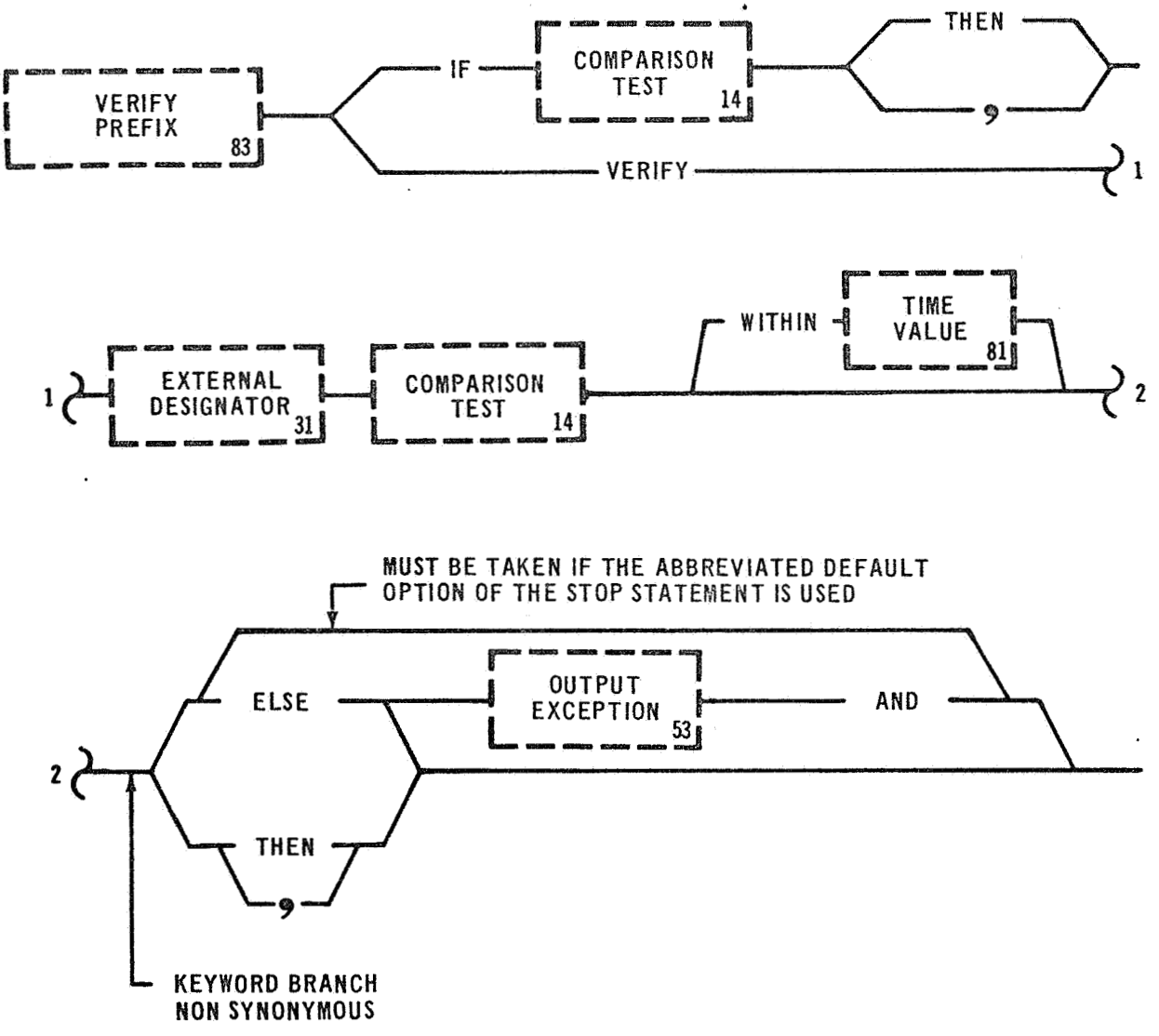
The Time Prefix permits a statement to be executed dependent upon a specific time condition. The Function Designator referenced must be a Time Function Designator type.

If the Key word "AFTER" is used the statement following the Time Prefix will be executed when the time specified is less than the actual time reading. If the Key word "WHEN" is used, the statement following the Time Prefix will be executed when the time specified is equal to or less than the actual time reading.

This is not a concurrent prefix. The system will delay accordingly until the Time Prefix is satisfied.

83
REV 1

## VERIFY PREFIX



6.6.4 Verify Prefix

Examples: IF (A) = (B) THEN

VERIFY <SII CHILLDOWN VLV> IS CLOSED THEN

The Verify Prefix cannot be used as a stand alone statement, but may be used as an optional prefix to any Procedural Statement. The Verify Prefix provides a conditional transfer capability. The Verify Prefix has two main options: namely the "IF THEN" and "VERIFY" options.

The "IF THEN" option provides a conditional branching capability based upon comparison testing of internal data. The optional nature of the statement execution is provided by embedding in the "IF THEN" option a Relational Formula or Limit Formula for comparison testing. If the result of the evaluation of the formula is true, the remaining part of this statement will be executed. If the result of the evaluation of the formula is false, the remaining portion of the statement will not be executed, but the next statement in the written sequence will be executed.

The "VERIFY" option provides a conditional transfer capability based upon comparison testing of external data. The Verify option also uses the Limit Formula or Relational Formula for determining branching by the same method as the "IF THEN" option.

The "VERIFY" option also provides a negative comparison testing with the "ELSE" path. In this instance, if the result of the evaluation of the formula is false, the Output Exception unit will be executed next and followed by the rest of the statement. If the result of the evaluation of the formula is true, the next statement in the written sequence will be executed. The "VERIFY" option also provides

a time interval for testing. The result of the time interval test can be considered to be logically added with the result of the evaluation of the formula. If a time interval is specified and the condition of the comparison test is met before or as the time interval expires the result is regarded as true. If the time interval expires and the condition of the comparison test is not met, the result is considered false. Branching is then done is accordance with the "THEN" or "ELSE" options as described above. The word "THEN" and the symbol "," (comma) have the same meaning in the statement, when related to branching.

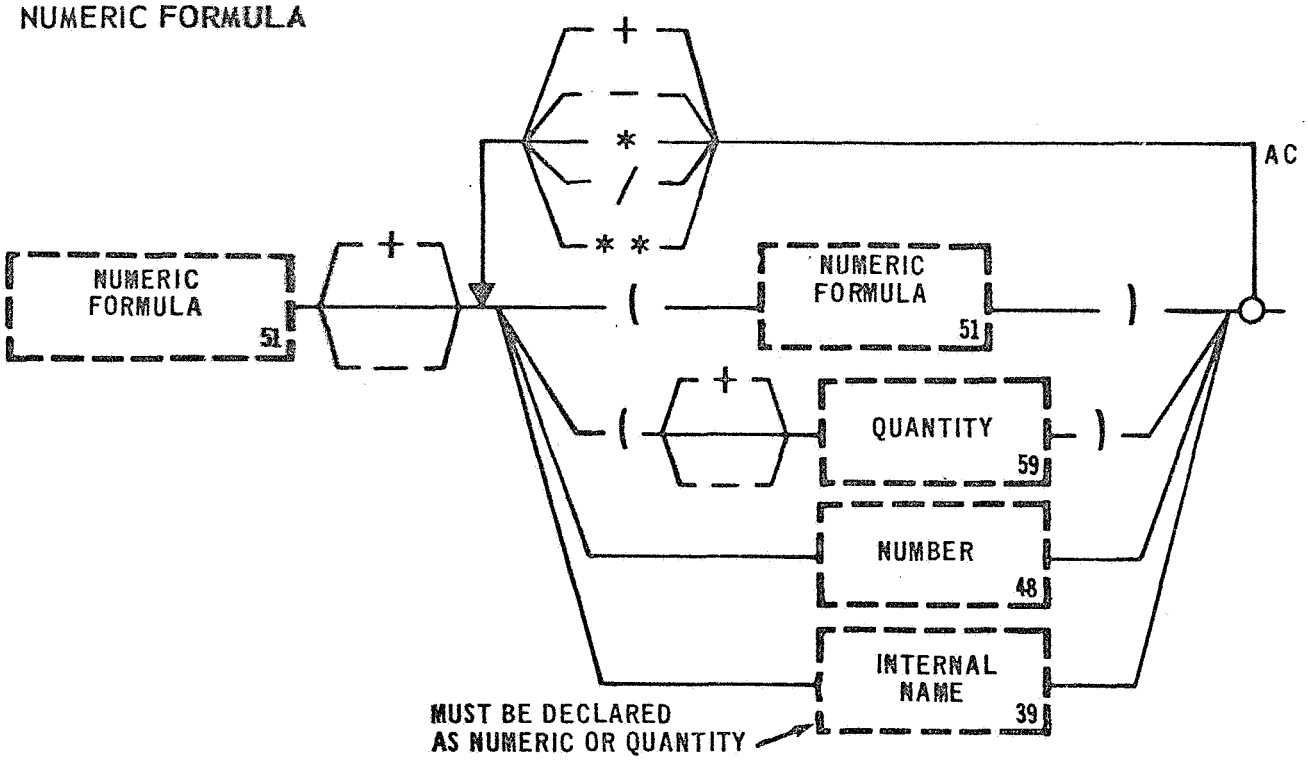
Examples: IF (TABLE A) COLUMN 3 ARE BETWEEN 80 AND 100, CLOSE
<LOX VENT VALVE 1> ;
VERIFY <D030-323> IS BETWEEN 1670 PSIA AND 1510 PSIA
ELSE DISPLAY EXCEPTION (D030-323 EXCEEDS REDLINE)
TO <CRT 2> AND TURN ON <CUTOFF> ;
VERIFY (TABLE A) FUNCTIONS ARE BETWEEN COLUMN 6 AND (TABLE
B) COLUMN 4;
VERIFY (TABLE C) FUNCTIONS ARE OFF;

## 6.7 FORMULAS

These elements include syntactical units defining the mathematical capabilities and comparison testing included in the language.

51
REV 0

## NUMERIC FORMULA



6.7.1 Numeric Formula

Examples: A+B+C/D**3

(ALPHA) * (BETA)

(-18 VOLTS) + (63 VOLTS) **2

- (63*(10 AMPS))

The Numeric Formula allows a mathematical operation to be performed on the right hand side of an equation. The mathematical symbols and their meanings are as follows:

MULTIPLICATION DENOTED BY ONE ASTERISK (*).

DIVISION DENOTED BY A SLASH (/).

ADDITION DENOTED BY A PLUS (+).

SUBTRACTION DENOTED BY A MINUS (-).

EXPONENTIATION DENOTED BY TWO ASTERISKS (**).

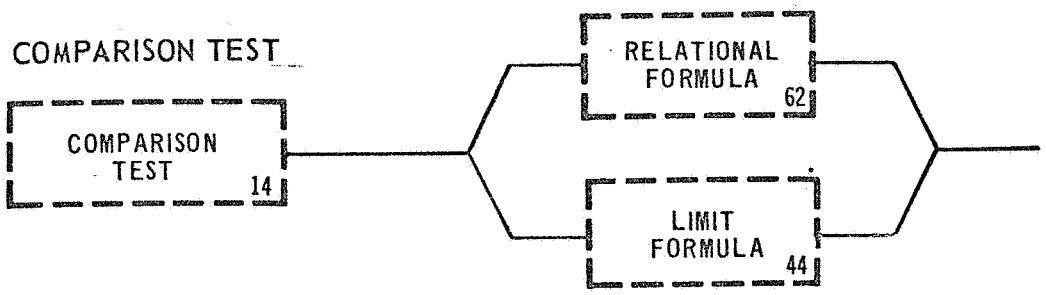
The following rules will be used in performing mathematical operations:

1. Parenthesis will be used where required to indicate order in which calculations are to be performed. Arithmetic operations within the innermost parenthesis are accomplished first.
2. When the heirarchy of operations in an expression is not completely specified by the use of parenthesis, the sequence reduction is as follows:
   (a) First, all exponentiations are performed.
   (b) Next, all multiplications and divisions are performed.
   (c) Finally, all additions and subtractions are performed.

Within a sequence of consecutive multiplications and/or subtractions in which the order of the operations is not completely specified by parenthesis, the meaning is that of a left to right reduction.

14
REV J

COMPARISON TEST

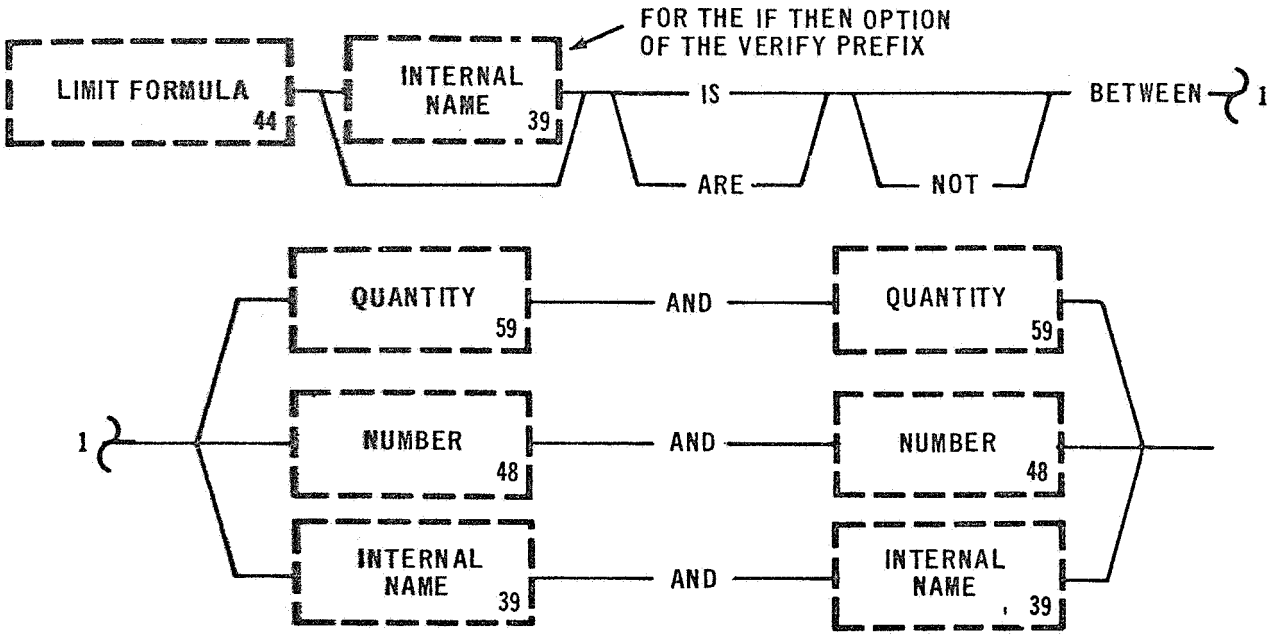


### 6.7.2 Comparison Test

The Comparison Test syntax diagram represents a logical grouping of the Relational Formula and Limit Formula syntax diagrams. Since the Comparison Test diagram is used in several other diagrams, this type of grouping results in simpler appearing diagrams.

44
REV 0

## LIMIT FORMULA



### 6.7.3 Limit Formula

Examples: IS BETWEEN 10 VOLTS AND 30 VOLTS

IS NOT BETWEEN (ALPHA) AND (BETA)

(PRESSURE A) IS BETWEEN 14 PSIA AND 20 PSIA

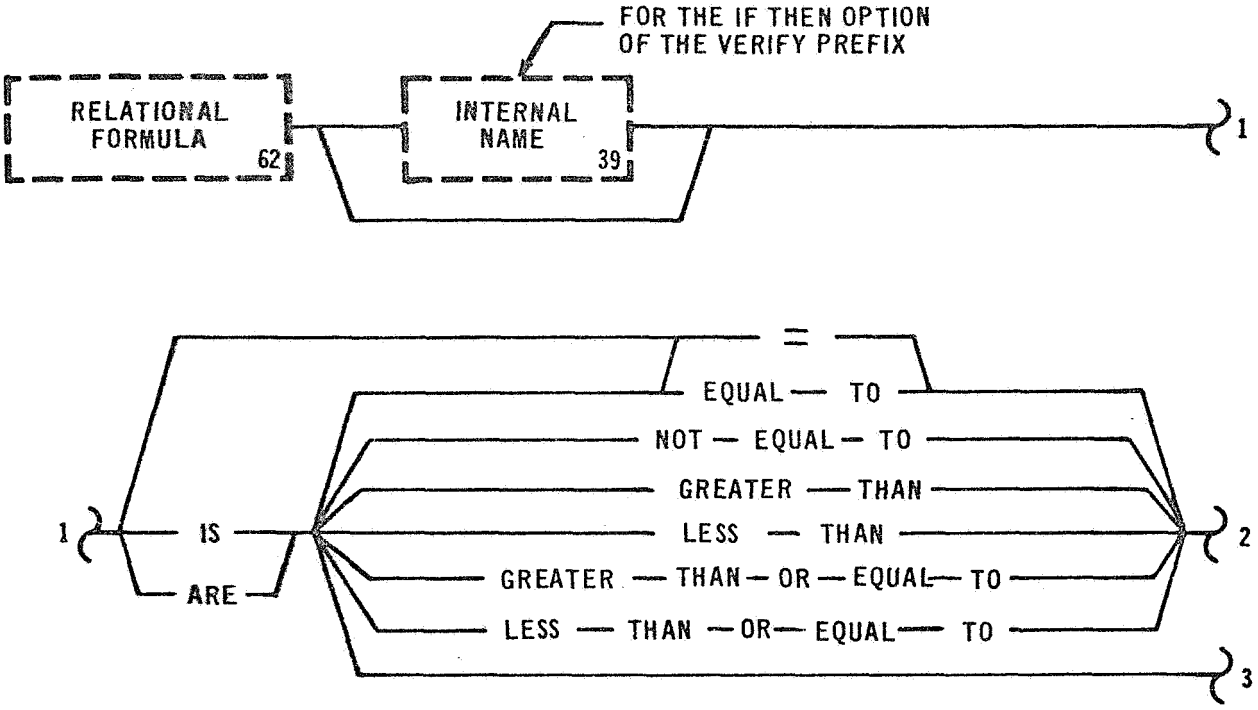
ARE NOT BETWEEN (HI) AND (LO)

The Limit Formula allows a comparison to be made between two limits. The comparison may be for either Internal Names or Function Designators. Function Designators will be written for the statement calling the Limit Formula and not in the Limit Formula itself. The Internal Name must be listed as part of the Limit Formula.

Caution: Incompatible data and variables will be the responsibility of the person preparing the program.

62
REV 0

# RELATIONAL FORMULA



6.7.4 Relational Formula

Examples: (PRESS A) = 10 PSIA

(PRESS A) IS EQUAL TO (PRESS B)

ARE ON

(PRESS A) IS GREATER THAN 15 PSIA

(BUS READING) IS LESS THAN 14 VOLTS

(BUS READING) IS GREATER THAN OR EQUAL TO 16 VOLTS

(BUS READING) IS LESS THAN OR EQUAL TO 24 VOLTS

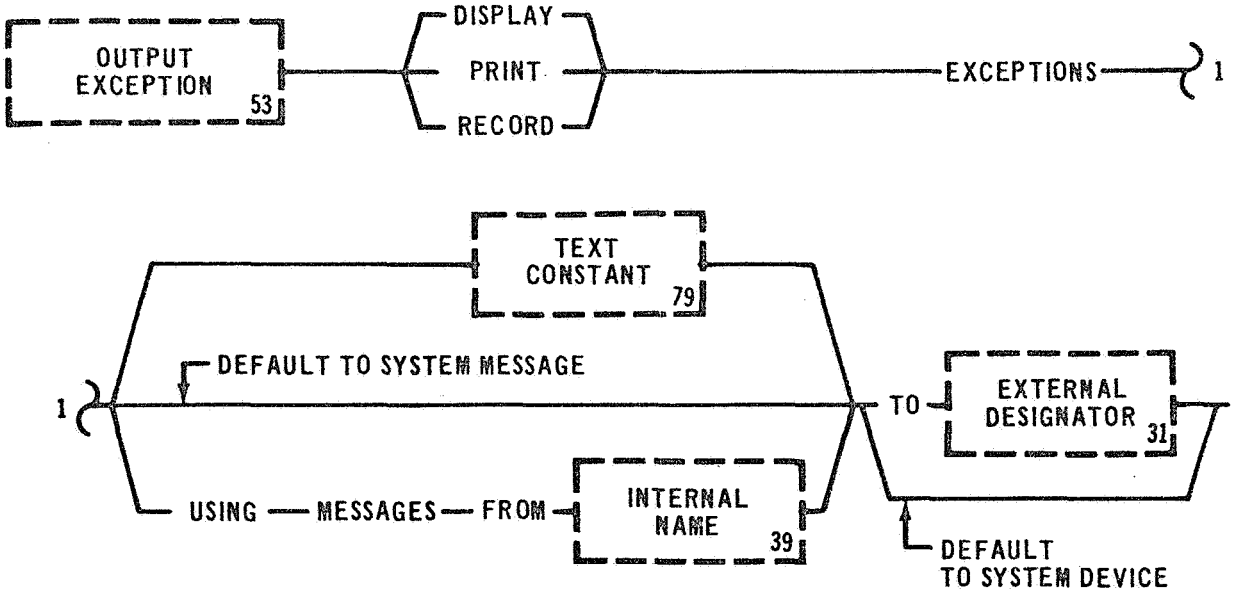
The Relational Formula allows for a verification to be made dependent upon a particular condition or state. Function Designators and Internal Names may be related to states or various data types. The sense of the relation may be either equality, greater or less than equality, or both. As with the Limit Formula if a Function Designator is being related then it must be listed from the statement referencing the Relational Formula and not the Relational Formula itself. The Internal Name is listed in the Relational Formula.

## 6.8 EXCEPTION

The Exception Group is made up of the Output Exception element. The Output Exception element is used in the Verify Prefix and the CONCURRENT STATEMENT.

53 REV 1

## OUTPUT EXCEPTION



6.8.1 Output Exception

Examples: PRINT EXCEPTIONS USING MESSAGES FROM (MESSAGE LIST) TO

<LINE PRINTER 1>

DISPLAY EXCEPTIONS

RECORD EXCEPTIONS TO <DISC>

The Output Exception is used to output messages to any recording device when an error condition occurs. This option allows for the use of system error messages, if any, and also a default to a standard system recording device, if any.

If a list of messages is used in relation to a group of Function Designators, then they must correspond exactly in the number of entries. If an error condition occurs, then an error message will be chosen from the list in a one-to-one correspondence.

The Output Exception unit may be abbreviated when used in combination with the STOP STATEMENT to effect a Language Processor default condition in which the computer to engineer communication devices and to predefine the message to be indicated on the device. An example of the abbreviated form is:

VERIFY <LOX VENT VALVE 2> IS OFF;

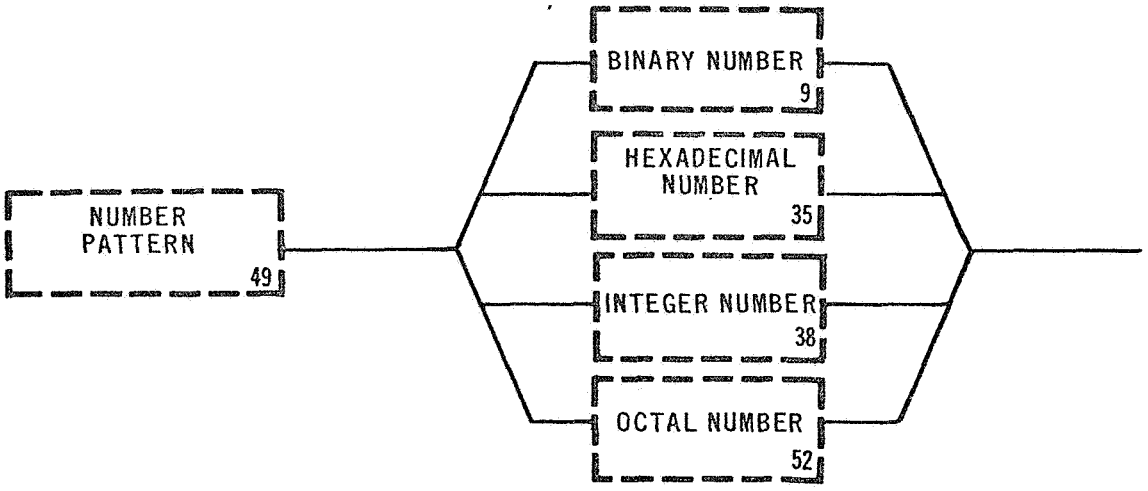
This statement has the same meaning as:

VERIFY <LOX VENT VALVE 2> IS OFF ELSE RECORD EXCEPTIONS AND STOP;

Reference the description of the STOP STATEMENT for more information on page 85.

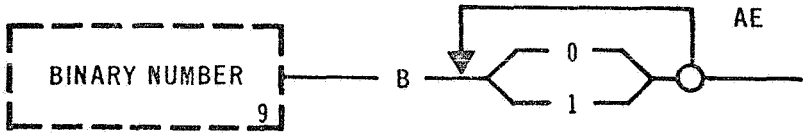
49
REV 0

## NUMBER PATTERN



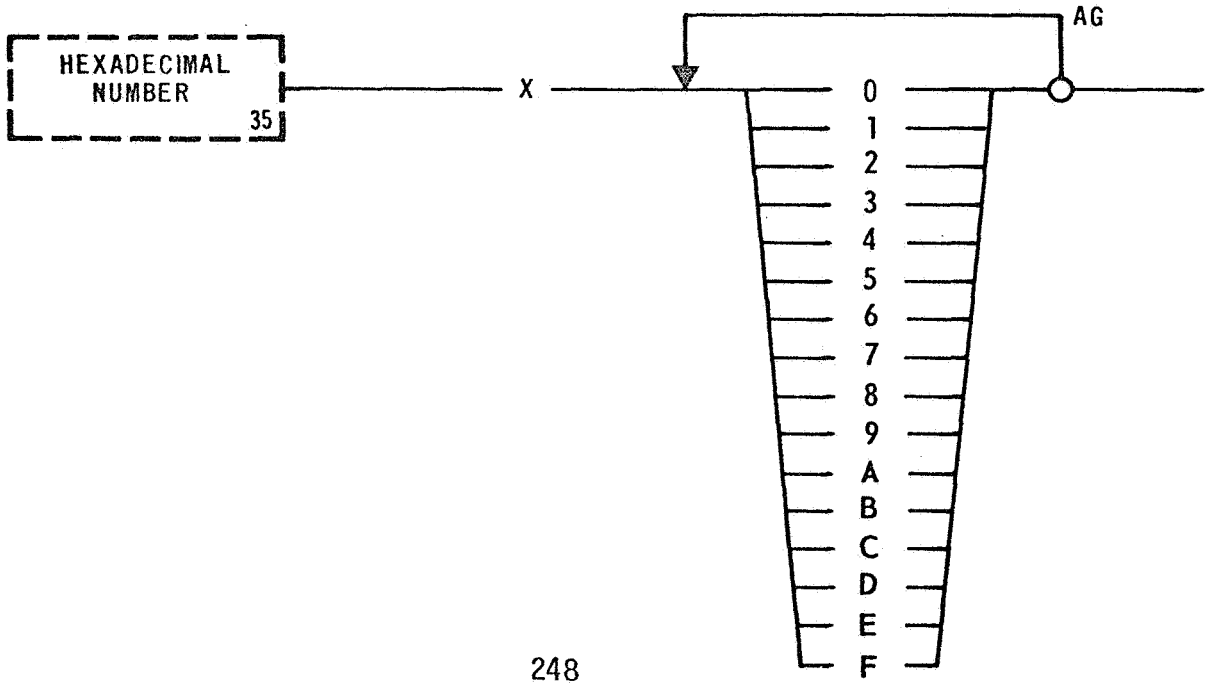

9
REV 0

## BINARY NUMBER




35
REV 0

## HEXADECIMAL NUMBER



## 6.9 NUMBER REPRESENTATION

These elements define the basic numeric character sets that are available to the test writer.

### 6.9.1 Number Pattern

The syntactical unit Number Pattern defines the types of numbers allowed. These include Binary, Octal, Decimal, and Hexadecimal.

### 6.9.2 Binary Number

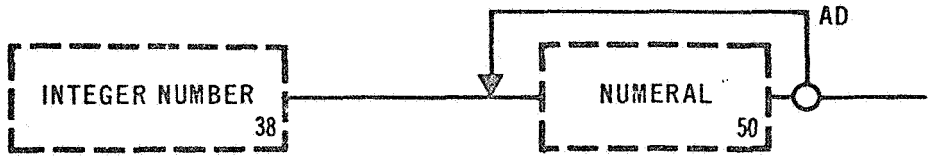
A Binary Number must begin with the letter "B" and is formed by any combination of the Numerals "1" and "0".

### 6.9.3 Hexadecimal Number

A Hexadecimal Number must begin with the letter "X" and may be formed by any combination of the Numerals "0" through "9" and the Letters A, B, C, D, E, or F.

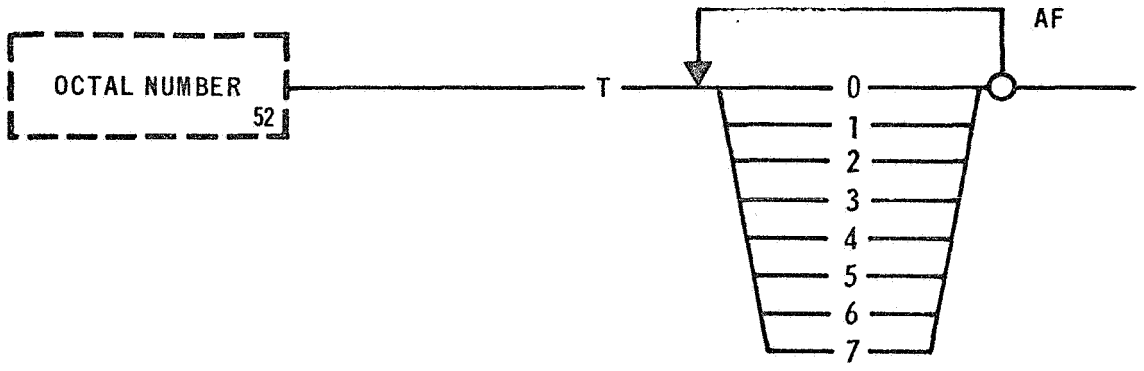
38
REV 0

## INTEGER NUMBER



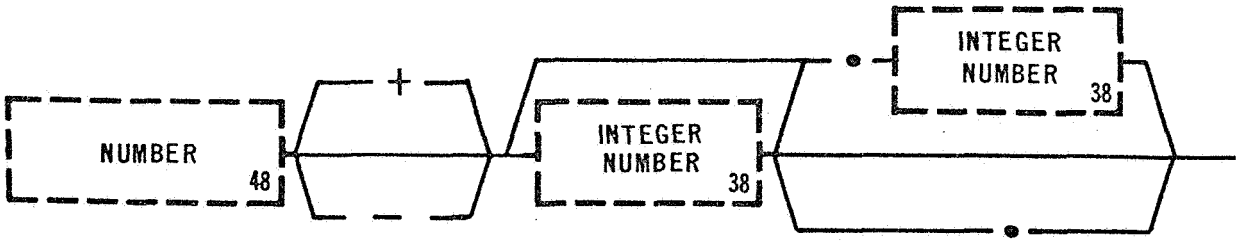

52
REV 1

## OCTAL NUMBER




48
REV 1

## NUMBER



6.9.4 Integer Number

An Integer Number is any combination of allowed numerals to form a decimal whole number.

6.9.5 Octal Number

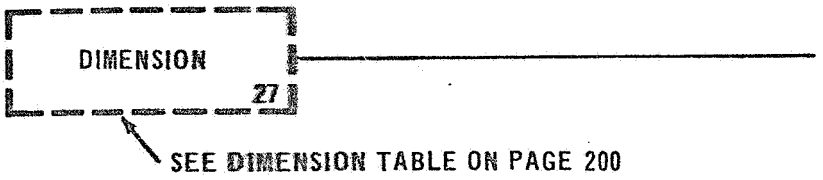
An Octal Number must begin with the letter "T" and may be formed by any combination of Numerals "0" through "7".

6.9.6 Number

A Number is a positive or negative Integer Number or fractional number. A Number may be written as 3.3 or 3 or 0.3 or .83.

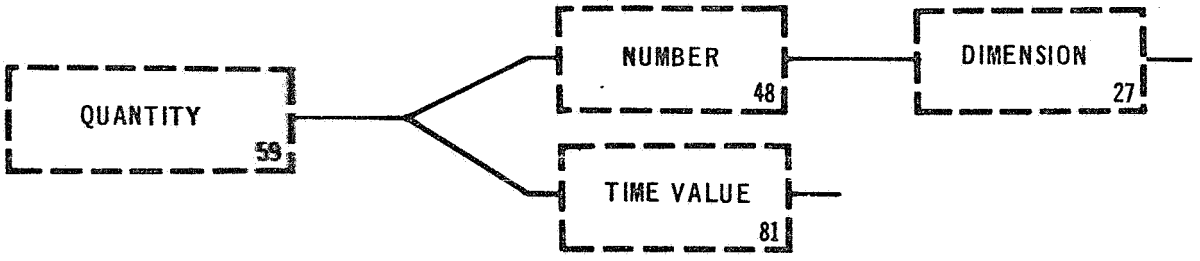
27
REV 0

## DIMENSION



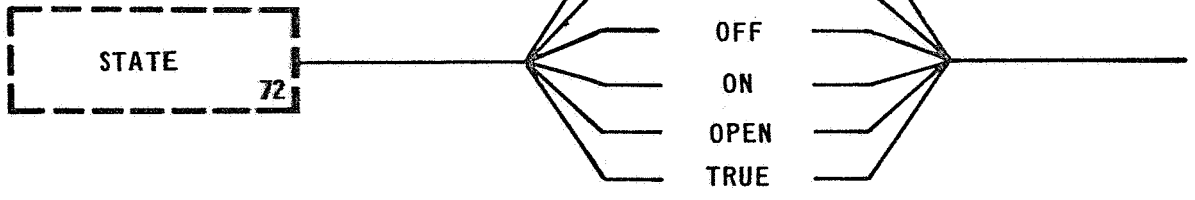

59
REV 0

## QUANTITY




72
REV 1

## STATE



## 6.10 Engineering Values (Dimensions)

The engineering units available are contained on the referenced page by the DIMENSION diagram.

### 6.10.1 Quantity

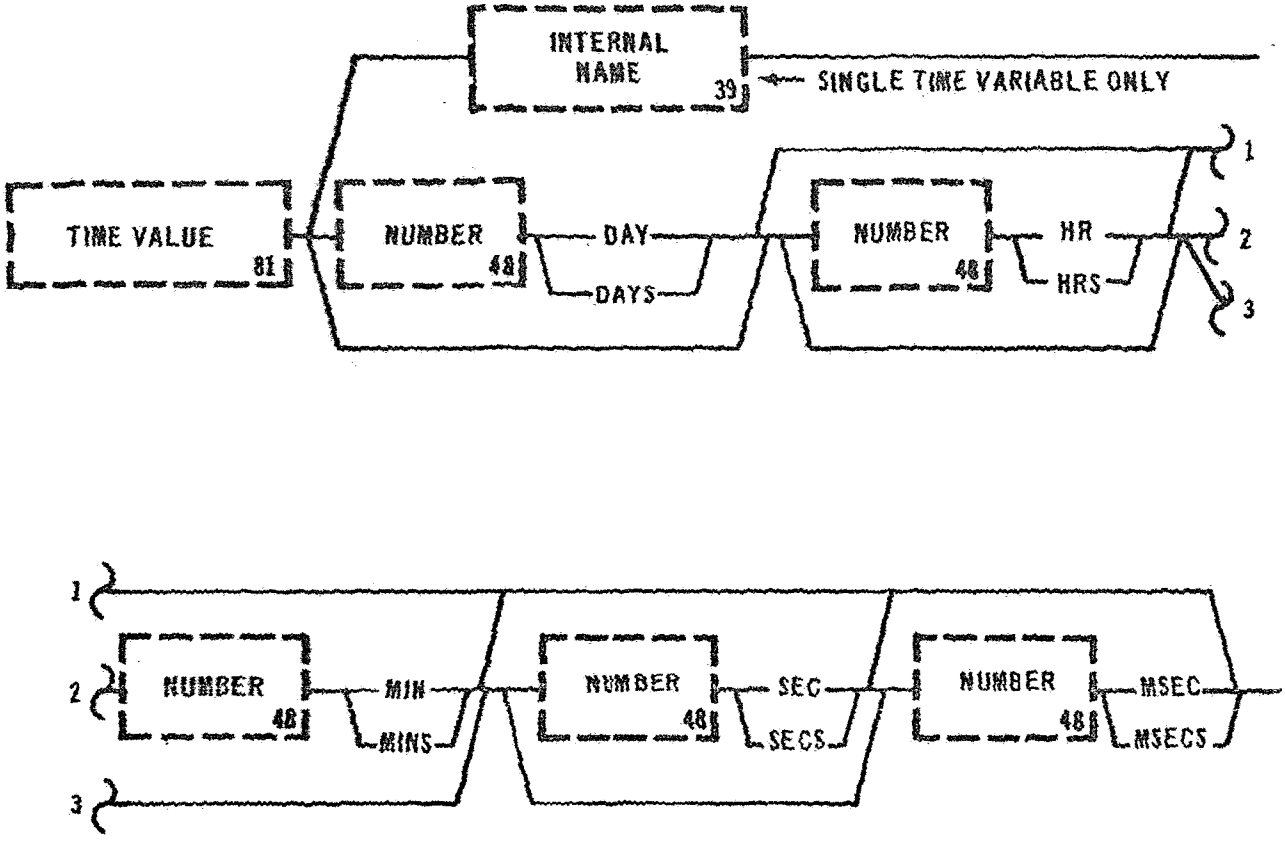
A Quantity is a number, with a Dimensional identifier attached, which provides both scaling information to the Language Processor and a method of error checking the test writer's use of these quantities.

### 6.10.2 State

A State represents a boolean expression. The allowable expressions are ON, OFF, OPEN, CLOSED, TRUE, or FALSE. ON, CLOSED, and TRUE are represented by a binary "1" and OFF, OPEN, and FALSE are represented by a binary "0".

81
REV 1

## TIME VALUE



### 6.10.3 Time Value

Examples: 3 DAYS

4 HRS 5 MINS 30 SECS

10 MINS 30 SECS

15 MINS

2 DAYS 20 HRS 10 MINS 22 SECS 15 MSECS

5.6 MSECS

(TIME)

The Time Value allows for numerical representation of DAYS, HRS, MINS, SECS, and MSECS. The Time Value may be positive or negative. Only the first entry in the Time Value should be signed. The Time Value also allows an Internal Name to be used.

7.0 CAPABILITY CHART

| STATEMENT NAME | TYPE | KEYWORD | STATEMENT CAPABILITY |
|---|---|---|---|
| 1. ACTIVATE TABLE | P | ACTIVATE | ALLOWS OPERATIONS TO BE PERFORMED ON PARTICULAR ROWS IN A TABLE. |
| 2. APPLY ANALOG | P | APPLY<br>SEND | APPLIES ANALOG VALUES TO THE SYSTEM UNDER TEST. |
| 3. ASSIGN | P | ASSIGN | PERFORMS LOGICAL OPERATIONS ON "STATE" (ON/OFF) INTERNAL PROGRAM DATA. |
| 4. AVERAGE | P | AVERAGE | PROVIDES A CONVENIENT MEANS OF AVERAGING DATA FROM A SYSTEM UNDER TEST. |
| 5. BEGIN DATA BANK<br>6. BEGIN MACRO<br>7. BEGIN PROGRAM<br>8. BEGIN SUBROUTINE | S | BEGIN | INITIAL BOUNDARY FOR DESCRIBED ITEMS. |
| 9. COMMENT | S | $ | ALLOWS NON-EXECUTABLE INFORMATION TO BE INSERTED FOR CLARIFICATION. |
| 10. CONCURRENT | P | CONCURRENTLY | ALLOWS CONCURRENT PROGRAM EXECUTION, MONITORING AND RECORDING. |
| 11. DECLARE DATA<br>12. DECLARE NUMERIC LIST<br>13. DECLARE NUMERIC TABLE<br>14. DECLARE QUANTITY LIST<br>15. DECLARE QUANTITY TABLE<br>16. DECLARE STATE LIST<br>17. DECLARE STATE TABLE<br>18. DECLARE TEXT LIST<br>19. DECLARE TEXT TABLE | D | DECLARE | DEFINES DATA CHARACTERISTICS NECESSARY FOR COMPILATION AND ENTERS INITIAL DATA VALUES. |
| 20. DELAY | P | DELAY | CAUSES A PROGRAM HOLD FOR A SPECIFIC TIME INCREMENT OR UNTIL A PARTICULAR EVENT OCCURS. |

TYPE

D - DECLARATION STATEMENT
P - PROCEDURAL STATEMENT
S - SYSTEM STATEMENT

| STATEMENT NAME | TYPE | KEYWORD | STATEMENT CAPABILITY |
|---|---|---|---|
| 21. DISABLE INTERRUPT | P | DISABLE | INHIBITS THE EXECUTION OF A SUBROUTINE ON THE OCCURRENCE OF A SPECIFIED INTERRUPT. |
| 22. END | S | END | FINAL BOUNDARY FOR GOAL COMPONENTS. |
| 23. EXPAND MACRO | S | EXPAND EXECUTE | A COMPILER DIRECTIVE THAT EXPANDS THE MACRO AND ALSO PROVIDES CONTROL OVER THE PRINTING OF THE MACRO DURING COMPILATION. |
| 24. FREE DATA BANK | S | FREE | CANCELS AN ACTIVE DATA BANK USED BY THE LANGUAGE PROCESSOR. |
| 25. GO TO | P | GOTO | PROVIDES FOR UNCONDITIONAL BRANCHING. |
| 26. INHIBIT TABLE | P | INHIBIT | INHIBITS OPERATIONS ON PARTICULAR ROWS OF A TABLE. |
| 27. ISSUE DIGITAL PATTERN | P | ISSUE | ISSUES A DIGITAL PATTERN TO THE SYSTEM UNDER TEST. |
| 28. LEAVE | S | LEAVE | ALLOWS AN EXIT FROM GOAL INTO A ROUTINE WRITTEN IN ANOTHER LANGUAGE LOCATED IN A DATA BANK SUBROUTINE. |
| 29. LET EQUAL | P | LET | PROVIDES MATHEMATICAL OPERATIONS OF ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION, AND EXPONENTIATION. |
| 30. PERFORM PROGRAM<br>31. PERFORM SUBROUTINE | P | PERFORM | CALLS DESCRIBED PROGRAM/SUBROUTINE INTO EXECUTION. UPON THE ITEM'S TERMINATION, CONTROL IS RETURNED TO THE NEXT STATEMENT AFTER THE PERFORM STATEMENT. |
| 32. READ | P | READ MEASURE | MEASURES A PARTICULAR FUNCTION AND SAVES THE RESULT. |
| 33. RECORD DATA | P | RECORD DISPLAY PRINT | OUTPUTS DATA TO COMPUTER/ENGINEER COMMUNICATION DEVICES. |

| STATEMENT NAME | TYPE | KEYWORD | STATEMENT CAPABILITY |
|---|---|---|---|
| 34. RELEASE CONCURRENT | P | RELEASE | STOPS CYCLIC EXECUTION OF THOSE ITEMS INITIATED BY A CONCURRENT STATEMENT. |
| 35. REPEAT | P | REPEAT | ALLOWS ONE OR MORE STATEMENTS TO BE REPEATED A SPECIFIED NUMBER OF TIMES. |
| 36. REPLACE | S | REPLACE | A WRITER AID TO ALLOW REPLACEMENTS OF NAMES OR ABBREVIATED CODING. |
| 37. REQUEST KEYBOARD | P | REQUEST | ALLOWS DATA TO BE ENTERED INTO A PROGRAM. |
| 38. RESUME | S | RESUME | FINAL BOUNDARY FOR A DATA BANK SUBROUTINE TO USE ANOTHER LANGUAGE. |
| 39. SET DISCRETE | P | SET<br>OPEN<br>CLOSE<br>TURN ON<br>TURN OFF | SENDS A DISCRETE (ON/OFF) COMMAND TO THE SYSTEM UNDER TEST. |
| 40. SPECIFY | S | SPECIFY | PROVIDES THE DATA TO INTERFACE THE TEST PROCEDURE TERMINOLOGY WITH THE SYSTEM UNDER TEST. |
| 41. STOP | P | STOP | STOPS PROGRAM EXECUTION UNTIL MANUALLY RESTARTED. |
| 42. TERMINATE | P | TERMINATE | CAUSES TERMINATION OF PROGRAM OR SUBROUTINE. |
| 43. USE DATA BANK | S | USE | ACTIVATES A DATA BANK FOR USE BY THE LANGUAGE PROCESSOR. |
| 44. WHEN INTERRUPT | P | WHEN | ACTIVATES AN INTERRUPT AND INDICATES A SUBROUTINE TO BE EXECUTED WHEN THE INTERRUPT OCCURS. |

## 7.1 GOAL KEYWORD/PHRASE INDEX

| KEYWORD/PHRASE | STATEMENT TITLE | DIAGRAM NO. |
|---|---|---|
| Entries | Declare Quantity List Statement | 20 |
| | Declare State List Statement | 22 |
| | Declare Text List Statement | 24 |
| Entry From | Request Keyboard Statement | 66 |
| Equal To | Assign Statement | 3 |
| | Declare Data Statement | 17 |
| | Let Equal Statement | 42 |
| | Relational Formula | 62 |
| Every | Concurrent Statement | 15 |
| Exceptions | Output Exception | 53 |
| Execute | Expand Macro Statement | 30 |
| Expand | Expand Macro Statement | 30 |
| False | State | 72 |
| For | Repeat Statement | 64 |
| | Set Discrete Statement | 70 |
| Free | Free Data Bank Statement | 32 |
| Functions | External Designator | 31 |
| Goto | When Interrupt Statement | 84 |
| Go To | Goto Statement | 34 |
| Greater Than | Relational Formula | 62 |
| Greater Than or Equal To | Relational Formula | 62 |
| Hr | Time Value | 81 |
| Hrs | Time Value | 81 |
| If | Verify Prefix | 83 |
| Inhibit | Inhibit Table Statement | 37 |
| Is | Limit Formula | 44 |
| | Relational Formula | 62 |
| | Time Prefix | 80 |
| Issue | Issue Digital Pattern Statement | 40 |
| Leave | Leave Statement | 41 |
| Less Than | Relational Formula | 62 |
| Less Than or Equal To | Relational Formula | 62 |
| Let | Let Equal Statement | 42 |
| Load | Specify Statement | 71 |
| Macro | Begin Macro Statement | 6 |
| | End Statement | 29 |
| Measure | Read Statement | 60 |
| Msec | Time Value | 81 |
| Msecs | Time Value | 81 |
| Min | Time Value | 81 |
| Mins | Time Value | 81 |
| Not | Limit Formula | 44 |
| Not Equal To | Relational Formula | 62 |
| Number | Declare Data Statement | 17 |
| Occurs | When Interrupt Statement | 84 |
| Off | State | 72 |
| On | State | 72 |
| Open | Set Discrete Statement | 70 |
| | State | 72 |

| KEYWORD/PHRASE | STATEMENT TITLE | DIAGRAM NO. |
|---|---|---|
| Or Until | Delay Statement | 26 |
| Perform | Concurrent Statement | 15 |
| | Perform Program Statement | 55 |
| | Perform Subroutine Statement | 56 |
| Present Value Of | Apply Analog Statement | 2 |
| | Concurrent Statement | 15 |
| | Issue Digital Pattern Statement | 40 |
| | Record Data Statement | 61 |
| | Set Discrete Statement | 70 |
| Print | Concurrent Statement | 15 |
| | Record Data Statement | 61 |
| Program | Begin Program Statement | 7 |
| | Concurrent Statement | 15 |
| | End Statement | 29 |
| Quantity | Declare Data Statement | 17 |
| Read | Read Statement | 60 |
| Readings Of | Average Statement | 4 |
| Record | Concurrent Statement | 15 |
| Release | Release Concurrent Statement | 63 |
| Repeat | Repeat Statement | 64 |
| Replace | Replace Statement | 65 |
| Request | Request Keyboard Statement | 66 |
| Resume | Resume Statement | 67 |
| Revision | Revision Label | 68 |
| Row | Activate Table Statement | 1 |
| | Inhibit Table Statement | 37 |
| Rows And | Declare Numeric Table Statement | 19 |
| | Declare Quantity Table Statement | 21 |
| | Declare State Table Statement | 23 |
| | Declare Text Table Statement | 25 |
| Sec | Time Value | 81 |
| Secs | Time Value | 81 |
| Send | Apply Analog Statement | 2 |
| Sensor | Specify Statement | 71 |
| Set | Set Discrete Statement | 70 |
| Specify | Specify Statement | 71 |
| State | Declare Data Statement | 17 |
| Step | Step Number | 73 |
| Stop and Indicate Restart Labels | Stop Statement | 74 |
| Subroutine | Begin Subroutine Statement | 8 |
| | End Statement | 29 |
| | Perform Subroutine Statement | 56 |
| System | Specify Statement | 71 |
| Terminate | Terminate Statement | 78 |
| Text | Declare Data Statement | 17 |
| | Leave Statement | 41 |
| | Perform Subroutine Statement | 56 |
| | Record Data Statement | 61 |
| | Request Keyboard Statement | 66 |

| KEYWORD/PHRASE | STATEMENT TITLE | DIAGRAM NO. |
|---|---|---|
| Then | Verify Prefix | 83 |
| | Time Prefix | 80 |
| Through | Repeat Statement | 64 |
| Times | Repeat Statement | 64 |
| To | Apply Analog Statement | 2 |
| | Concurrent Statement | 15 |
| | Issue Digital Pattern Statement | 40 |
| | Output Exception | 53 |
| | Record Data Statement | 61 |
| | Set Discrete Statement | 71 |
| True | State | 72 |
| Turn Off | Set Discrete Statement | 70 |
| Turn On | Set Discrete Statement | 70 |
| Type | Specify Statement | 71 |
| Use | Use Data Bank Statement | 82 |
| Using | Specify Statement | 71 |
| Using Messages From | Output Exception | 53 |
| Verify | Concurrent Statement | 15 |
| | Verify Prefix | 83 |
| Wait | Delay Statement | 26 |
| When | Time Prefix | 80 |
| When Interrupt | When Interrupt Statement | 84 |
| With | Declare Numeric List Statement | 18 |
| | Declare Quantity List Statement | 20 |
| | Declare Quantity Table Statement | 21 |
| | Declare State List Statement | 22 |
| | Declare Text List Statement | 24 |
| | Declare Text Table Statement | 25 |
| | Replace Statement | 65 |
| With A Maximum of | Declare Data Statement | 17 |
| | Declare Text Table Statement | 25 |
| With Entries | Declare Numeric List Statement | 18 |
| | Declare Quantity Table Statement | 21 |
| | Declare State Table Statement | 23 |
| | Declare Text Table Statement | 25 |
| Within | Verify Prefix | 83 |

## 7.2 GOAL STATEMENT INDEX

| NO. | GOAL STATEMENT NAME | TYPE | PAGE | DIAGRAM NUMBER | KEYWORD/PHRASE |
|---|---|---|---|---|---|
| 1. | ACTIVATE TABLE | P | 121 | 1. | ACTIVATE |
| 2. | APPLY ANALOG | P | 47 | 2. | APPLY/SEND |
| 3. | ASSIGN | P | 93 | 3. | ASSIGN |
| 4. | AVERAGE | P | 63 | 4. | AVERAGE |
| 5. | BEGIN DATA BANK | S | 129 | 5. | BEGIN DATA BANK |
| 6. | BEGIN MACRO | S | 135 | 6. | BEGIN MACRO |
| 7. | BEGIN PROGRAM | S | 131 | 7. | BEGIN PROGRAM |
| 8. | BEGIN SUBROUTINE | S | 133 | 8. | BEGIN SUBROUTINE |
| 9. | COMMENT | S | 155 | 13. | $ |
| 10. | CONCURRENT | P | 99 | 15. | CONCURRENTLY/EVERY/ PERFORM/DISPLAY/VERIFY |
| 11. | DECLARE DATA | D | 11 | 17. | DECLARE DATA |
| 12. | DECLARE NUMERIC LIST | D | 15 | 18. | DECLARE NUMERIC LIST |
| 13. | DECLARE NUMERIC TABLE | D | 27 | 19. | DECLARE NUMERIC TABLE |
| 14. | DECLARE QUANTITY LIST | D | 17 | 20. | DECLARE QUANTITY LIST |
| 15. | DECLARE QUANTITY TABLE | D | 31 | 21. | DECLARE QUANTITY TABLE |
| 16. | DECLARE STATE LIST | D | 19 | 22. | DECLARE STATE LIST |
| 17. | DECLARE STATE TABLE | D | 35 | 23. | DECLARE STATE TABLE |
| 18. | DECLARE TEXT LIST | D | 21 | 24. | DECLARE TEXT LIST |
| 19. | DECLARE TEXT TABLE | D | 39 | 25. | DECLARE TEXT TABLE |
| 20. | DELAY | P | 75 | 26. | DELAY |
| 21. | DISABLE INTERRUPT | P | 117 | 28. | DISABLE |
| 22. | END | S | 137 | 29. | END |
| 23. | EXPAND MACRO | S | 157 | 30. | EXPAND/EXECUTE |
| 24. | FREE DATA BANK | S | 149 | 32. | FREE |
| 25. | GOTO | P | 79 | 34. | GO TO |
| 26. | INHIBIT TABLE | P | 125 | 37. | INHIBIT |
| 27. | ISSUE DIGITAL PATTERN | P | 51 | 40. | ISSUE |
| 28. | LEAVE | S | 139 | 41. | LEAVE |
| 29. | LET EQUAL | P | 95 | 42. | LET |
| 30. | PERFORM PROGRAM | P | 105 | 55. | PERFORM PROGRAM |
| 31. | PERFORM SUBROUTINE | P | 109 | 56. | PERFORM SUBROUTINE |
| 32. | READ | P | 67 | 60. | READ/MEASURE |
| 33. | RECORD DATA | P | 59 | 61. | RECORD |
| 34. | RELEASE CONCURRENT | P | 103 | 63. | RELEASE |
| 35. | REPEAT | P | 81 | 64. | REPEAT |
| 36. | REPLACE | P | 161 | 65. | REPLACE |
| 37. | REQUEST KEYBOARD | P | 71 | 66. | REQUEST |
| 38. | RESUME | S | 143 | 67. | RESUME |
| 39. | SET DISCRETE | P | 55 | 70. | SET/OPEN/CLOSE/ TURN ON/TURN OFF |
| 40. | SPECIFY | S | 151 | 71. | SPECIFY |
| 41. | STOP | P | 85 | 74. | STOP |
| 42. | TERMINATE | P | 89 | 78. | TERMINATE |
| 43. | USE DATA BANK | S | 147 | 82. | USE |
| 44. | WHEN INTERRUPT | P | 113 | 84. | WHEN/OCCURS |

TYPE: D- DECLARATION STATEMENT, P- PROCEDURAL STATEMENT, S- SYSTEM STATEMENT

## 7.3 GOAL ELEMENTS INDEX

## 7.4 FEEDBACK LETTERS VERSUS DIAGRAM CHART

| LETTER | PROPOSED VALUE | DIAGRAM NAME |
|---|---|---|
| A | | Declare Data Statement |
| B | | Declare Data Statement |
| C | | Record Data Statement<br>Request Data Statement |
| D | | Activate Table Statement<br>Inhibit Table Statement |
| E | | Apply Analog Statement<br>Issue Digital Pattern Statement<br>Set Discrete Statement |
| F | | Leave Statement<br>Perform Subroutine Statement |
| G | | Release Concurrent Statement |
| H | | Stop Statement |
| I | | Disable Interrupt Statement |
| J | | Begin Macro Statement<br>Expand Macro Statement |
| K | | Begin Subroutine Statement |
| L | | Free Data Bank Statement<br>Use Data Bank Statement |
| M | | Specify Statement |
| N | | Specify Statement |
| P | | Character String |
| R | | Function Designator |
| S | | External Designator |
| T | | Name |
| W | | Macro Label |
| Y | | Revision Label |

FEEDBACK LETTERS VERSUS DIAGRAM CHART (CONTINUED)

| LETTER | PROPOSED VALUE | DIAGRAM NAME |
|---|---|---|
| AB | | Step Number |
| AC | | Numeric Formula |
| AD | | Integer Number |
| AE | | Binary Number |
| AF | | Octal Number |
| AG | | Hexadecimal Number |

## 7.5 INDEX OF SYNTAX DIAGRAMS